A Nested U-Net Architecture for Stenosis Segmentation

Hongyu Chen
Department of Mathematics
University of Waterloo
20792778
h542chen@uwaterloo.ca

Marcus Huang
Department of Mathematics
University of Waterloo
21061943
m43huang@uwaterloo.ca

INTRODUCTION

The paper introduces UNet++, a novel architecture built on the original U-net by transforming the connectivity between encoder and decoder sub-networks through the redesigned skip pathways. The main idea behind UNet++ is to bridge the semantic gap between the feature maps of the encoder and decoder prior to fusion. In the original U-Net, the feature maps from the encoder are directly received by the decoder.

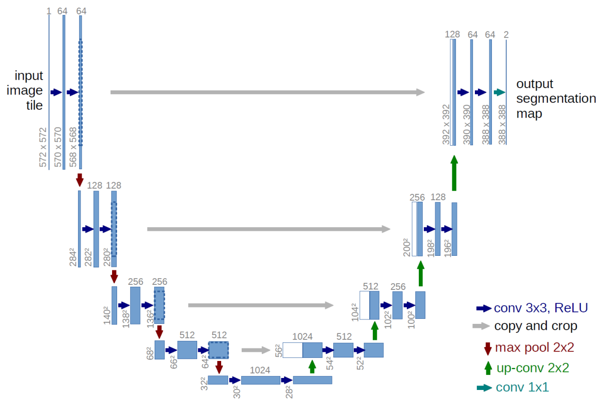

Figure 1: UNet Structure

However, in UNet++, these feature maps will first pass through a dense convolution block whose number of layers depends on the pyramid level, thereby bringing the semantic level of the encoder feature maps near to those in the decoder. A concatenation layer precedes each layer in these dense blocks. These layers fuse the output from the prior convolution layer with the up-sampled output of the lower dense block. This new design aims to make it easier for the optimizer by ensuring the feature maps are semantically similar. Furthermore, the skip connections are normally plain in U-Net. These skip connections are replaced with nested dense skip pathways in UNet++. Let $x^{i,j}$ denote the output of the node $X^{i,j}$, where i indexes the down-sampling layer long the encoder and j indexes the convolution layer within the dense block along the skip pathway. **[Figure 2]** The stack of feature maps represented by $x^{i,j}$ is computed as follow:

$$x^{i,j} = \begin{cases} \mathcal{H}\left(x^{i-1,j}\right), & j = 0 \\ \mathcal{H}\left(\left[\left[x^{i,k}\right]_{k=0}^{j-1}, \mathcal{U}(x^{i+1,j-1})\right]\right), & j > 0 \end{cases}$$

Where the function $H(\cdot)$ represents a convolution operation followed by an activation function, $U(\cdot)$ denotes an up-sampling layer, and [ ] indicates the concatenation of feature maps. Another significant improvement is UNet++ includes deep supervision, which involves training the model with multiple output layers to enhance performance and enable model pruning at inference time. The following equation presents the combined loss function we used, similar to the one in the deep supervision approach. The loss function integrates both Binary Cross-Entropy (BCE) and the Dice loss to enhance segmentation accuracy. Binary Cross-Entropy loss is used to ensure stable and effective training, while Dice loss addresses class imbalance between the stenosis and background regions.

$$\mathcal{L} = \mathcal{L}^{BCE} + \mathcal{L}^{dice}$$
$$= -(Y\log\hat{Y} + (1-Y)\log(1-\hat{Y})) + (1 - \frac{2Y \cap \hat{Y}}{Y \cup \hat{Y}})$$

The authors claim that the UNet++ structure changes collectively bridge the semantic gap between the encoder and decoder feature maps. It improves gradient flow and maintains model accuracy while allowing for potential reductions in model size.

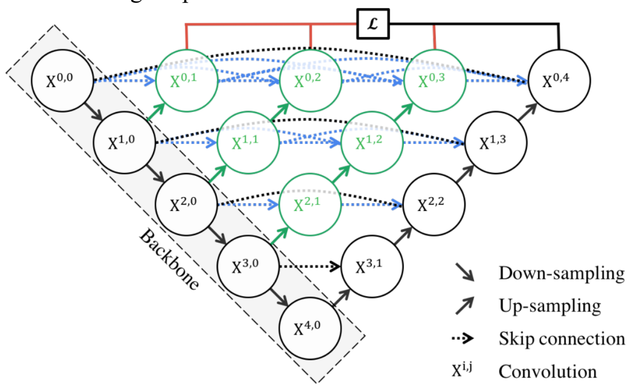

Figure 2: UNet++ Structure

DATASET

To replicate the results and further extend the methods of the original paper, we introduced a different dataset to apply the methods on another task in the same domain.

Coronary artery disease (CAD) is a condition caused by the buildup of atherosclerotic plaque in the coronary arteries, leading to restricted blood flow to the heart. It is a leading cause of death worldwide. Coronary angiography is the primary diagnostic tool for CAD, using contrast material and X-rays to visualize artery lesions and blood flow in real-time. This allows for accurate detection of stenosis and supports decisions about interventions like stent insertions.

However, segmenting coronary arteries and detecting stenotic lesions is a time-consuming and costly process when done manually. Automated tools for lesion detection and localization could improve the speed and accuracy of CAD diagnosis and treatment.

Automatic Region-based Coronary Artery Disease diagnostics using the X-ray angiography imagEs (ARCADE) curated a dataset consisting of two sets of 1500 XCA images for two of the supervised tasks in the challenge: coronary artery segmentation and stenosis detection. In this project we will mainly focus on stenosis detection. The results are evaluated using the mean F1 score obtained on a set of 300 test images for each of the tasks.

EXPERIMENT

Baseline Model

We compared Unet++ to the original U-Net and a custom wide U-Net with a similar parameter count. U-Net was chosen as a common segmentation baseline, while the wide variant ensured any performance gains weren't solely due to more parameters.

Model	$X^{0,0}$ / $X^{0,4}$	$X^{1,0}$ / $X^{1,3}$	$X^{2,0}$ / $X^{2,2}$	$X^{3,0}$ / $X^{3,1}$	$X^{4,0}$ / $X^{4,0}$
U-Net	32	64	128	256	512
Wide U-Net	35	70	140	280	560

Table 1: Number of convolutional kernels

Implementation Detail

We have largely replicated the original UNet++ architecture, with the additional experiments of replacing the VGG blocks with ResNet-style residual blocks and ResNeXt bottleneck blocks. However, due to computational resource limitations, the results obtained from these modifications were not satisfactory. As such, these changes are only included in the source code for reference. Additionally, we implemented post-processing techniques from a separate study, which involve counting the areas of connected components in the model's output and filtering out small blobs. This approach was explored to assess potential improvements in inference-time processing and enhance the overall accuracy of the model's predictions.

Result

Table 3 compares U-Net, wide U-Net and UNet++ in terms of parameters count and performance for the task of stenosis detection. Wide U-Net improves over U-Net due to its higher parameter count. UNet++ without deep supervision significantly outperforms both, indicating the benefit of its advanced architecture. Adding deep supervision provides a slightly boost.

Architecture	Params	F1	IoU
U-Net	7.77M	49.36	36.76
Wide U-Net	9.29M	53.52	39.37
UNet++ w/o DS	9.16M	57.47	41.30
UNet++ w/ DS	9.16M	57.76	41.68

Table 2: Segmentation results for U-net, Wide U-Net, UNet++

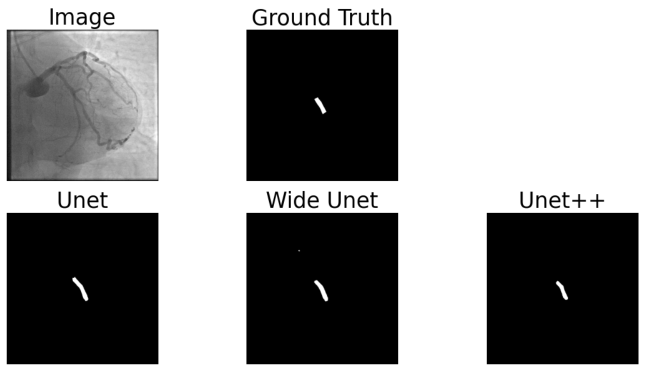

Figure 3: Qualitative comparison between U-Net, wide U-Net, and UNet++

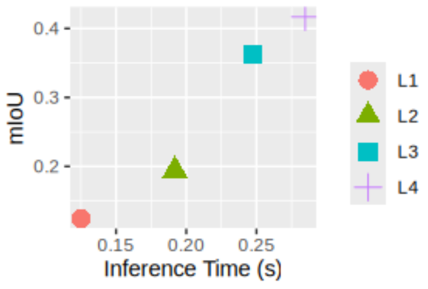

Figure 4: Complexity, speed, and accuracy of UNet++ after pruning on the test dataset

Model Pruning

We also replicated the model pruning experiments on the new dataset (refer to Fig. 5 for details). Compared to the original results, the accuracy degradation on the new dataset is more pronounced, even with only one level pruned. While pruning one level reduced inference time by 13.24%, it also led to a 12.91% drop in accuracy. This suggests that pruning may not be as effective on this particular dataset.

CONCLUSION

To replicate and build upon the results presented in the original UNet++ paper, we re-implemented the network following the outlined specifications and conducted additional experiments beyond the original methodology. The results indicate that UNet++ outperforms both U-Net and Wide U-Net on a new dataset, validating the claims made in the original paper and demonstrating enhanced performance.

Appendix

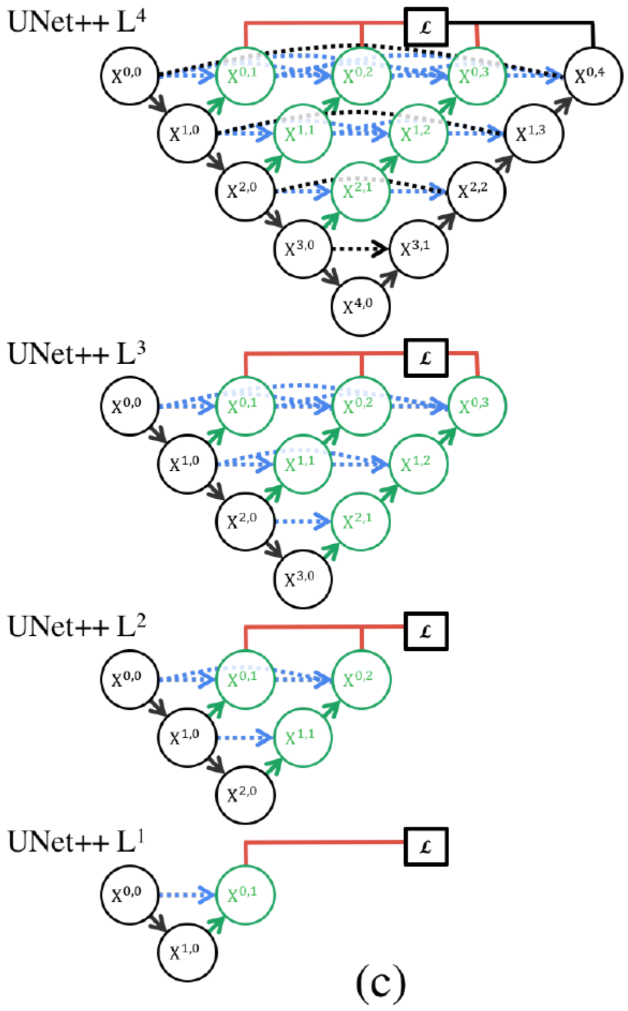

Figure 5: UNet++ Structure

REFERENCES

[1] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. 2018. UNet++: A Nested U-Net Architecture for Medical Image Segmentation. Retrieved from https://arxiv.org/abs/1807.10165

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. Retrieved from https://arxiv.org/abs/1505.04597

[3] Hui Lin, Tom Liu, Aggelos Katsaggelos, and Adrienne Kline. 2023. StenUNet: Automatic Stenosis Detection from X-ray Coronary Angiography. arXiv preprint arXiv:2310.14961 (2023).

[4] Aladdin Persson. 2020. Machine Learning Collection. GitHub. Retrieved from https://github.com/aladdinpersson/Machine-Learning-Collection/blob/master/ML/Pytorch/image_segmentation/semantic_segmentation_unet/train.py

[5] Pau Rodriguez. 2020. ResNeXt.pytorch. Github. Retrieved from https://github.com/prlz77/ResNeXt.pytorch/blob/master/models/model.py