Model Summary

Competition Name: SenNet + HOA - Hacking the Human Vasculature in 3D  
Team Name: ryo  
Private Leaderboard Score: 0.755959  
Public Leaderboard Score: 0.043141

# Background

I got my degree in “Bachelor of Arts (Japanese Language and Literature)” from Hosei University in Japan. And now, I have working for BIPROGY Inc. which changed its name from Nihon Unisys, Ltd. in 2022.

I have been working in business application development and have applied various methodologies and frameworks in practice. We business application developers value writing beautiful source code. However, I believe that machine learning source code written by academics, beauty is taken lightly. Smart people translate complex things into source code as they are complex, I think. They don't simplify it or focus on readability and maintainability. So, I decided to explore how to write beautiful machine learning code through a Kaggle competitions. I chose SenNet + HOA because it's an image competition; I avoided the natural language processing competition because my PC is poor.

After several days of programming, over 2 weeks of machine learning, and one week of tuning submissions, I gave up because my public score on the leaderboard was too low. And I got second place; I still can't believe it.

# Summary

* Model is U-Net 3D. Using focal loss and cosine decay scheduling
* Data augmentation using random rotations and positions
* Post-processing. Removing small positive chunks; positive chunks are searched by depth-first search.

# Model

I selected U-Net 3D for the deep learning model. (A colleague of mine is developing a program for rheumatoid arthritis diagnosis using numerical processing of x-ray images as input, and I thought that the knowledge might be useful when we convert the program to 3D in the future. The size of the U-Net 3D input was 128 x 128 x 32 dots. This size was not determined by looking at the data, but was adjusted to a size that was trainable within the memory of my GPU (NVIDIA GeForce RTX 4090).

Focal loss was used for the loss function. The ratio of blood vessels to the liver is very small, the ratio of true/false in data is asymmetric.

The learning rate was set using a cosine decay schedule with restarts. The large number of steps is required for the cosine decay schedule with restarts, but I judged to be acceptable because the amount of data increases with data augmentation, which is discussed below.

# Data

Considering the large amount of data required to train U-Net 3D, I first considered data augmentation. Since the data is 3D, any number of different data can be generated by simply changing the viewpoint. My program generates 10,000 data per kidney for every 4 epochs. The epoch size is 700, my program used 5.25 million data for training.

Data generation is realized by the same process as viewport conversion in 3D graphics. An affine transformation matrix was created from the camera information (eye position, viewing position, and upward direction), and the pixel positions in the 3D data were obtained by multiplying by the inverse matrix. Since the above process is time-consuming, I made maximum use of NumPy functions and parallelized them to speed up.

I also considered the risk of overfitting, since the data is originally the same even if the viewpoint is changed. Therefore, I used kidney\_1\_dense, kidney\_2, and kidney\_3\_sparse data to obtain generalization performance that would work with different patients and manufactures; although the published leaderboard score shows that it was not sufficient.

Because of the possibility of false-negatives when using sparse data, I considered curriculum learning, in which only dense data was used again after the training was completed, but decided against it because I did not think my PC performance would allow me to train U-Net 3D again within the competition period.

In addition, I used normalized data this time, but when I tuned the submission program to avoid runtime errors, I learned that clipping is effective. I considered re-starting over with standardized data, but decided against it because I did not think I would be able to complete the training in time.

# Post Processing

Since I trained with sparse data, I thought that many false negatives would be generated, and although I could reduce the false negatives by decreasing the threshold, this would increase the number of false positives. Therefore, I decided to search for connected true chunks, taking advantage of the fact that blood vessels are connected (otherwise, blood would not flow). At first, I programmed selecting the largest chunk, thinking that the accuracy of my predictor is high, but the public leaderboard score was 0, so I changed the method to remove the smallest chunks.

Since the method of selecting the largest chunk was effective in the private leaderboard score, I believe that the low accuracy in the public data was simply due to the lack of generalization performance of my predictor. If you have a predictor with high generalization performance, you may achieve high accuracy by searching for connected true chunks, and if there is an edge of another chunk near the edge of the chunk, you can connect them by assuming that there is a false negative between them, and if there is no edge of another chunk nearby, you can remove it by assuming that it is a false positive.

Connected true chunks were searched by using depth-first search. Implementing the search algorithm in plain Python is not good in performance, so if this post processing is actually useful, I would like to re-write it in a fast programming-language such as C++ or Rust.