Project report: TransUNet-based approach for Segmentation in Medical Images

Nguyen Thien Bao
Faculty of Information Technology
VNUHCM - University of Science
22127032@student.hcmus.edu.vn

Le Phuoc Phat
Faculty of Information Technology
VNUHCM - University of Science
22127322@student.hcmus.edu.vn

Vo Hoai Viet
Faculty of Information Technology
VNUHCM - University of Science
vhviet@fit.hcmus.edu.vn

***Abstract*—Medical image segmentation is a medical technique that enhances the accuracy of predictions in medical fields. Thanks to Transformer, which is designed for sequence-to-sequence prediction, many tasks about medical image segmentation achieve superior performance when using it on different medical applications, including multi-organ segmentation and cardiac segmentation. In the previous project proposal, we presented some technical knowledge about the topic of segmentation in medical images in general and TransUNet in particular, then we also proposed an approach and a plan to enhance the existing disadvantages of this model. In this project report, we will experiment, analyze, and evaluate the original TransUNet model on the Synapse multi-organ segmentation dataset [1]. Code and models are available at TransUNet Segmentation In Medical Image**

***Index Terms*—Segmentation in Medical Images, Deep Convolutional Neural Network, Vision Transformer, UNet**

I. Introduction

Medical image segmentation plays a huge role in many medical fields, especially in diagnostics, treatment planning, and disease monitoring. Segmenting medical images with high performance, such as MRI, CT, and X-ray scans, is essential for automated clinical segmentation. Among different CNN [2], [3] variants, UNet [4], which includes a symmetric encoder-decoder network with skip-connections to improve detail retention, has become one of the dominant approaches that has achieved many tremendous successes in a wide range of medical applications. Although this approach has a lot of exceptional representational power, it also exhibits limitations for modeling explicit long-range relations because of the intrinsic locality of convolution operations. To tackle these issues, many researchers studied self-attention mechanisms based on CNN characteristics. In this day and age, Transformer [5] has sparked innovations not only in the field of natural language processing (NLP) but also that of computer vision. Furthermore, Transformers are not only powerful at modeling global contexts but also show superior transferability for downstream tasks under large-scale pre-training. Thanks to these UNet and Transformer features, Chen et al. introduced the TransUNet model [4] for the first time in 2021, which explores the potential of transformers in the context of medical image segmentation.

In this report, we will explore the method of building the architecture of transformers. From that, we also analysed and experimented with the model on the dataset and evaluated the results. During this implementation, we may get a bad result compared to the original one because of the difficulty in the technical devices, such as the GPU and the lack of time.

II. Method

The objective of the segmentation task is to find the class that each pixel belongs to. The input of the task is an image $x \in R^{H \times W \times C}$, where H and W are the width and height of the image, and C is the number of channels. The output predicted by models is a matrix of size $H \times W$, known as the segmentation mask or segmentation map. Each element in the mask represents the class of the pixel in the input image. The segmentation task is basically the classification task, which CNNs have effectively solved recently. The difference in the classification task is the segmentation task requires the output as a map, with the resolution of the original input. Therefore, after extracting features using CNNs, instead of using them to make predictions, the goal is to use them to reconstruct the features back to the original spatial structure.

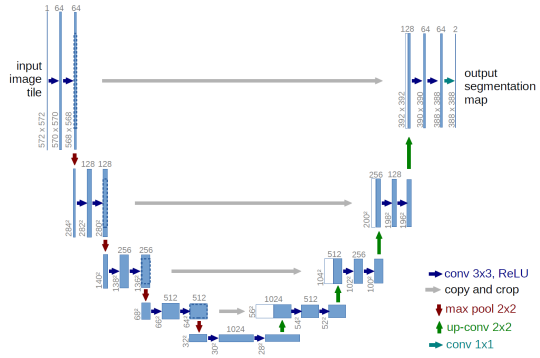

Fig. 1. U-Net architecture

The U-shape or V-shape architecture is one of the most common ways for segmentation tasks. This architecture consists of 2 parts: an encoder symmetrizing with a decoder. The encoder consists of a series of convolutional layers alternating with pooling layers. The convolutional layers in the encoder take responsibility for modifying the input image, highlighting the hidden features in the image, while the pooling layers reduce the size of the extracted feature map, hence helping to

reduce computational complexity and overfitting. The decoder, in contrast to the encoder, consists of a series of convolutional layers and unpooling layers. The convolutional layers in the decoder are responsible for reconstructing the output mask, while the unpooling layers increase the size of the mask and help to retrieve the original information. The architecture also uses the skip connection mechanism. In the encoder branch, the feature maps generated by the convolutional layers are stored, and then, in the decoder branch, these stored features are concatenated with the input of the convolutional layers with corresponding depth. This mechanism helps preserve the original information of the input image, preventing it from being lost during the feature extraction process, hence improving the performance of the model.

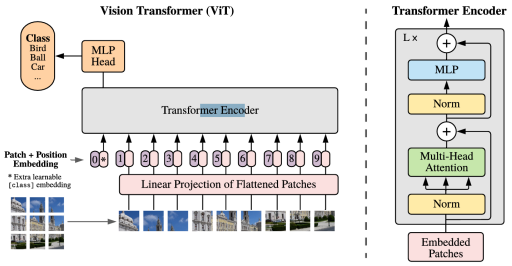

Fig. 2. Vision Transformer architecture

The TransUNet architecture is created by inserting a Vision Transformer architecture at the end of the encoder branch. The Vision Transformer architecture used in TransUNet consists of a linear projection layer, a patch and position embedding layer, an extra class embedding layer, and a series of 12 Transformer encoders. The linear projection layer divides the original input into many patches and presents them as vectors. The embedding layers take responsibility for storing feature vectors, position, and classification information. Each Transformer encoder layer consists of a multi-head attention layer, which helps to extract global relationships between image patches, and a multi-layer perceptron head to apply activation functions.

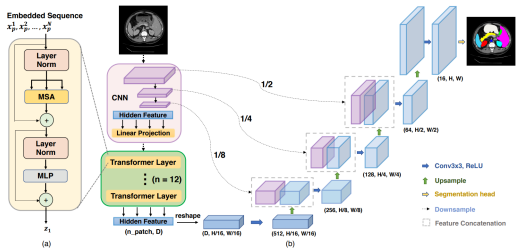

Fig. 3. TransUNet architecture

After extracting feature maps by the Vision Transformer, the model continues passing them through the decoder, applying convolutions, and concatenating with skip connections like the same with the original U-shape architecture.

III. Data

To evaluate and test the original TransUNet model, we will experiment with the benchmarks that these models used for their evaluation.

Synapse multi-organ segmentation dataset [1]: This dataset is about 50 abdominal CT scans of were randomly selected from a combination of an ongoing colorectal cancer chemotherapy trial, and a retrospective ventral hernia study. This data is labeled by 13 abdominal organs: spleen, right kidney, left kidney, gallbladder, esophagus, liver, stomach, aorta, inferior vena cava, portal vein and splenic vein, pancreas, right adrenal gland, and left adrenal gland.

In this project, we use the 30 abdominal CT scans, with 3779 axial contrast-enhanced abdominal clinical CT images in total. Each CT volume consists of approximately 85 to 198 slices of 512 x 512 pixels. We will train on 18 training cases that have a total of 2211 axial slices and test on 12 testing cases, each of which has been labelled on 8 abdominal organs (aorta, gallbladder, spleen, left kidney, right kidney, liver, pancreas, and stomach).

A. Training dataset

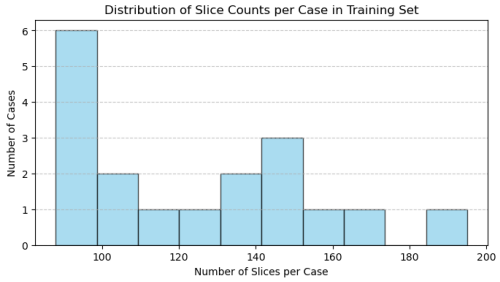

Fig. 4. Training Dataset

In the bar chart 4, we have 18 training cases where the minimum slices per case is 88 slices and the maximum ones are 195 slices. We can see that the training set has a non-uniform distribution of slices (most cases are 90 - 100, but many cases span up to 200 slices). This model needs to handle various levels of "thickness" (number of slices) of medical images.

B. Testing dataset

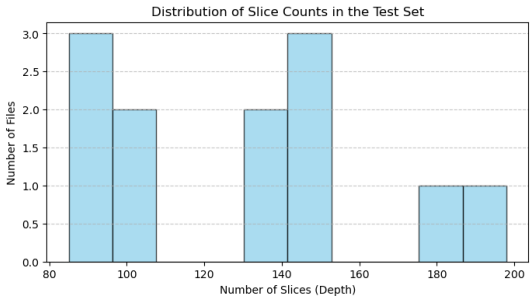

Fig. 5. Testing Dataset

In the testing dataset 5, we also have 12 testing cases where the maximum depth of the cases is 198 slices and the minimum depth is 85 slices. We can also observe that this test set is not uniform in terms of slice count; it has two main clusters, which are 80-100 and 120-140, with a few near 200 slices. This diversity can help validate the model's ability to generalize across various volume thickness during model development.

C. Visualization Samples

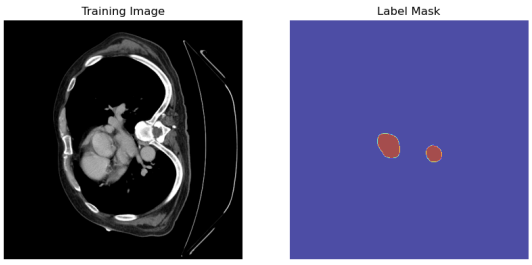

Fig. 6. Training case0005_slice115.npz sample

The training sample 6 is the image of case 0005 on the 155th slice.

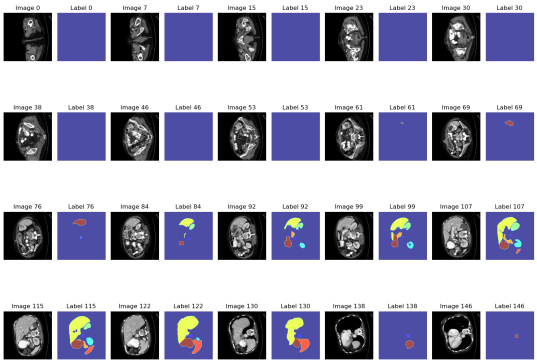

Fig. 7. Testing case0001.npy.h5 sample

The testing sample 7 is the picture of presenting all slices on the case 0001.

IV. Implementation

In this project, we mainly use PyTorch to implement the TransUNet model. Besides PyTorch, we also use libraries like Matplotlib, OpenCV, and SciPy for image processing.

A. Data Preprocessing

The first step is that you need to sign in to the official Synapse website and then download the raw dataset. Then, we need to set up the project directories and paths.

After that, we build the **SynapsePreprocessor** class, which provides methods to process the raw training and testing data for the Synapse dataset. This class will give a lot of methods: loading NIfTI files, clipping pixel values in the training images within a specified range from -125 to 275 and normalizing the values to the range [0, 1] using min-max normalization, resizing a 2D image slice to target shape using zoom from Scipy, splitting a 3D volume into 2D slices, saving processed data, processing training and testing data.

Finally, we implement the SynapseAugmentor for augmenting training images by randomly rotating them by multiples of 90 degrees and flipping them along one axis and also rotating the image by a random angle between -20 to 20 degrees. We also built the SynapseDataset class to support both training and testing splits.

B. Dataset Loader

The Synapse dataset stores images and masks in .npz format, each .npz file is a compression of an image and its mask. We implement a class named SynapseDataset inheriting the Dataset class of PyTorch, which helps to query the dataset more simply. The SynapseDataset requires a string, which is the path of the folder storing .npz files. The length of SynapseDataset is the number of files in the provided folder. Images and labels can be queried with the subscript operator. When the subscript operator is called, the SynapseDataset loads the .npz file at the provided index and returns a dictionary of image and mask and an extra field case name.

Before using the dataset for training, we implement some transformations like RandomGenerator and Zoomer. The RandomGenerator helps to augment the original data by randomly applying some modification. Each sample has a 1/3 chance to flip randomly, a 1/3 chance to rotate into a random direction, and a 1/3 chance to do nothing. The Zoomer helps to stretch the image into the correct size for the model to work properly. The dataset then is loaded in batches with a specific size using DataLoader from PyTorch.

C. Model architecture

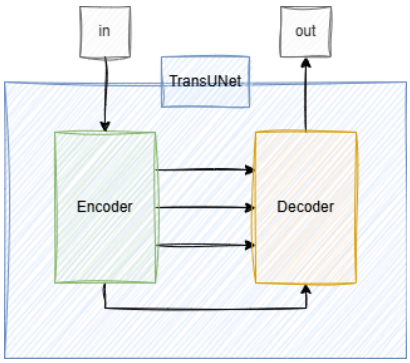

Fig. 8. The TransUNet module

The TransUNet module is the module for the whole architecture. This module consists of an Encoder and a Decoder module. The input image x passing through the Encoder becomes 4 parts: final extracted features x, skip connection features at each encoding stage x_1, x_2, and x_3. Then these 4 parts pass through the Decoder and get the final class score matrix of size $H \times W \times C$, where $H \times W$ is the size of the image and C is the number of classes including the background class.

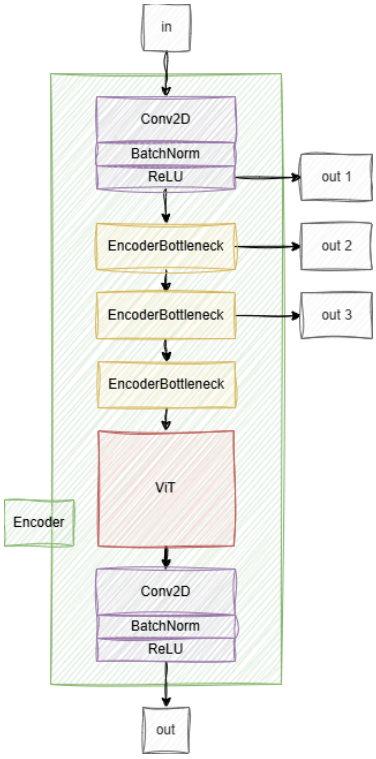

Fig. 9. The Encoder module

The Encoder module is the module for the encoder branch. This module consists of 2 convolutional modules, 3 EncoderBottleneck modules, and a ViT module. The input x passes through the first convolutional module, is batch normalized, and is activated by the ReLU function. Then it passes through 3 EncoderBottleneck modules. The features before passing x_1, x_2, and x_3 are stored for skip connection. The extracted features are passed to the ViT module and finally the second convolutional module.

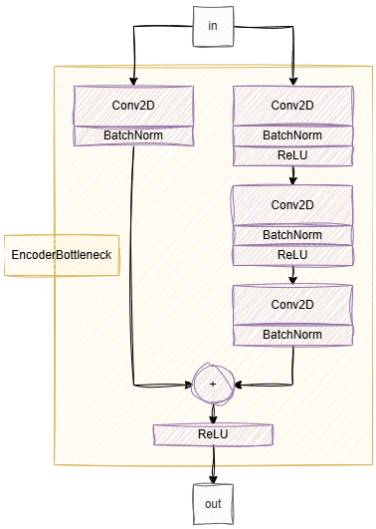

Fig. 10. The EncoderBottleneck module

The EncoderBottleneck module is located inside the encoder. This module divides the input into 2 flows: a flow passes through a convolutional layer with batch normalization, and another flow passes through several blocks that consist of a convolutional layer with batch normalization and is activated with the ReLU function. After that, both are merged with a plus operation and then activated with a ReLU function for the output.

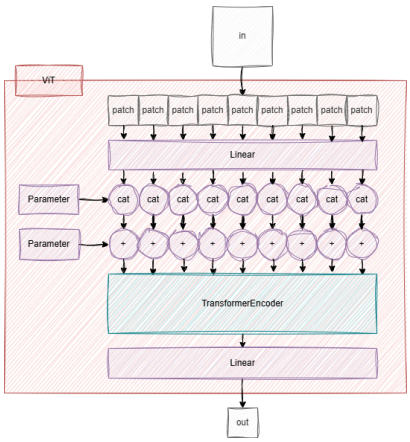

Fig. 11. The ViT module

After the EncoderBottleneck module extracts features, these features continue to pass down to the ViT module. In the ViT module, the input is divided into many small patches, and the linear projection layer presents them as vectors. Then, each patch is concatenated and augmented with extra embedding information. After that, these patches are fed to the transformer layers. Finally, the output of the Transformer layers passes through the MLP linear for classification (optional).

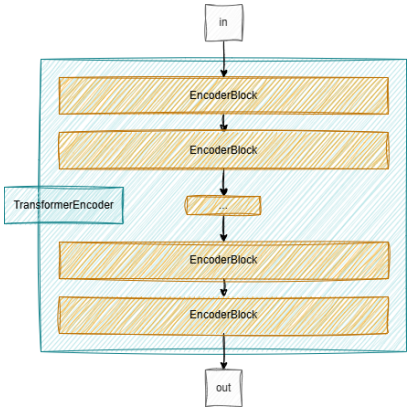

Fig. 12. The TransformerEncoder module

The TransformerEncoder module consists of 12 EncoderBlock modules. By repeating these EncoderBlock modules, the module can extract the features deeper with hierarchical representation.

The EncoderBlock module consists of a MultiHeadAttention module and 2 Linear layers. The output of the MultiHeadAttention module is combined with the original input by the

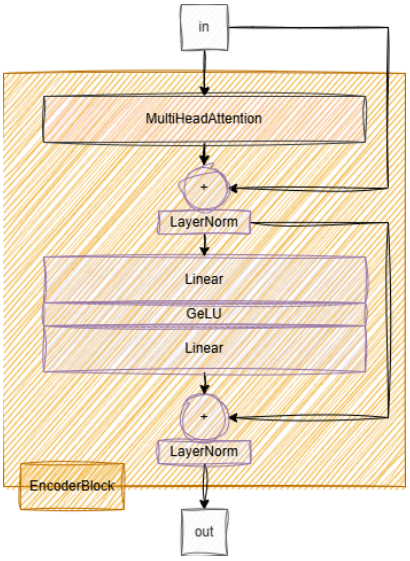

Fig. 13. The EncoderBlock module

plus operator and applies a layer normalization. After that, the flow passes down to the first linear layer, activated by the GeLu function, then the second linear layer. The output of the second linear layer continues to be combined with the output after layer normalization. Finally, apply one more layer normalization for the output.

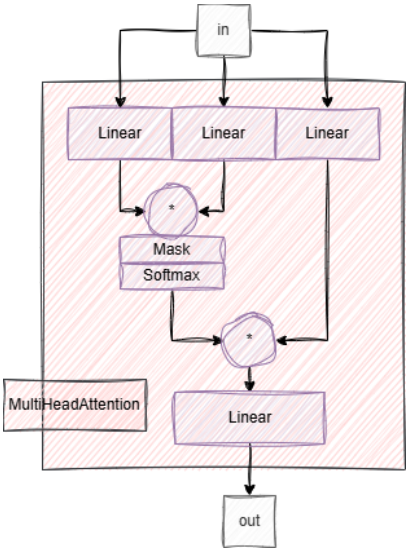

Fig. 14. The MultiHeadAttention module

The MultiHeadAttention module divides the flow into multiple heads. Each head continues to split the flow into 3 flows: query, key, and value. The query presents what the patch is looking for, the key presents the features of the patch for comparing with the query, and the value presents the actual feature information. Then the model combines the query with the key to present scores of which patches to give attention to. After that, the model continues combining the attention scores with the value for the final output.

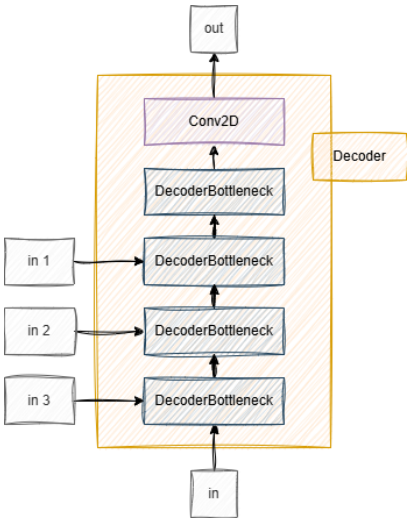

Fig. 15. The Decoder module

The Decoder module consists of 4 DecoderBottleneck modules. The input passing through 3 first DecoderBottleneck modules is combined with information from the skip connections. After that, the module passes input up to another DecoderBottleneck and finally applies the last $1x1$ convolutional layer to map the feature maps to the desired number of classes.

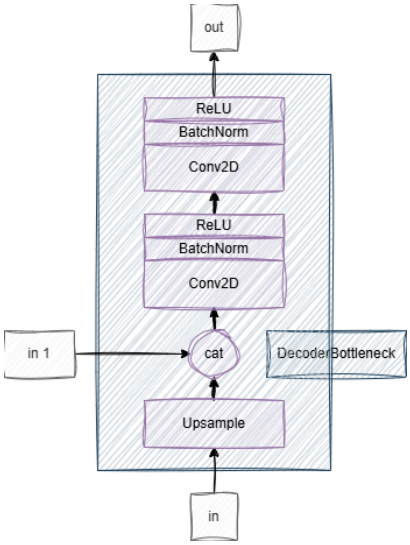

Fig. 16. The DecoderBottleneck module

The DecoderBottleneck module consists of 2 blocks of a convolutional layer with batch normalization and ReLU activation. First, the module concatenates the input with information from the skip connections. Then, the module passes up the concatenated input to these blocks to generate the mask while reconstructing the original spatial structure.

D. Training strategy

To train the model, firstly, we implement a class named ModelManager, this class takes responsibility for loading the pretrained model from the .pth file and performing a train or test step on a batch of input. Then, we implement a class named EpochCallback. An instance of EpochCallback is initialized at the beginning of the training process and is called at the end of each epoch. The EpochCallback checks training history, saves the model if its performance is better than the previous epoch, and also stops the training process if the performance does not improve by a specific number of epochs. We also implement a Logger class for logging the training history. The main class taking responsibility for the training process is the class Trainer. This class starts the training loop, performs training and testing, and then saves and monitors the history.

V. Experiment and Analysis

In this report, we conduct the main experiment on the Synapse multi-organ segmentation dataset by running the model with some specific hyperparameters. Particularly, we choose the number of output classes, which is 9 classes (including background and 8 organ classes). Besides, we set the input image size to 224x224 and use a batch size of 2 per GPU during training.

Because the model that we built from scratch requires a large amount of running time and computational resources, we decided to use the official version of the TransUNet model to do this experiment first.

The model is built upon a hybrid architecture that combines a ResNet-50 backbone with a Vision Transformer (ViT) block. In our implementation, the ViT configuration is set to R50-ViT-B_16 with a patch size of 16, meaning that the input image is divided into patches of 16 x 16 pixels.

To ensure reproducibility, we initialize the random seed to 1234 and set the training to be deterministic. We train the network for 5 epochs (or 2000 iterations in another configuration), using a base learning rate of 0.01.

The metrics used to evaluate the model's performance are the Dice coefficient and the 95% Hausdoff Distance (HD95)

Therefore, we will demonstrate the experimental results below:

A. Results Samples

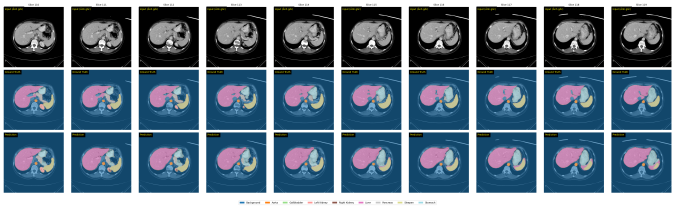

Fig. 17. Image

We provide the image result 17 after we test the TransUNet model on the Synapse multi-organ dataset.

In this test sample, we can see that the first row contains the original images of the patients, followed by the ground truth labels in the second row, and finally, the model's predicted results in the last row. These images are extracted from the original test set cases.

For this sample, the model provides mostly accurate predictions for organ identification. However, there are some errors in the later slices for the spleen and left kidney, indicating inconsistencies in segmentation for these organs.

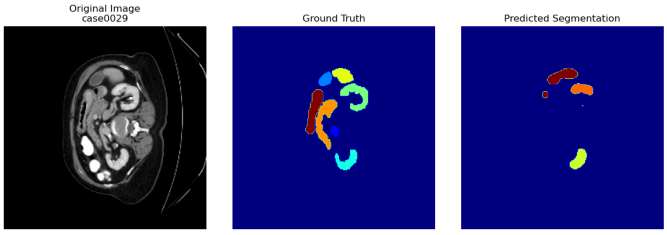

Fig. 18. Single Testing Sample

Figure 18 illustrates how the predicted segmentation changed compared to the ground truth images. The final result is not as good as the expected outcome.

B. Metrics

As we know, the dice score (ranging from 0 to 1) measures the overlap between prediction and ground truth. The dice score goes higher, the model's performance improves. In contrast, HD95 (95% Hausdorff Distance) measures boundary mismatch. The HD95 more lower, and the performance more better.

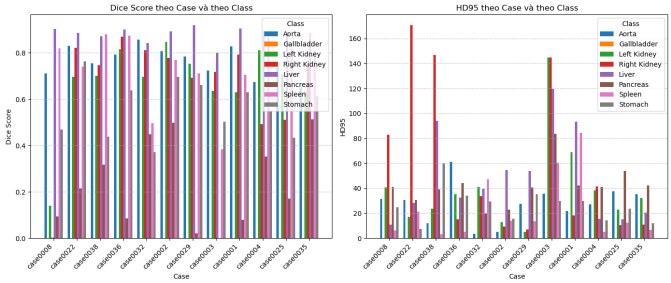

Fig. 19. Dice Score and HD95 by Case and by Classes

The above graphs 19 give information about two metrics that illustrate how well the model segments each organ in each class.

Overall, on the dice score chart, Liver, Aorta, and Spleen often stand out with higher scores in many testing samples, which indicates relatively good segmentation overlap for these organs, whereas Gallbladder frequently shows extremely low scores, suggesting the model struggles to detect it in most cases. In addition, on the HD95 chart, the Right Kidney sometimes presents the highest HD95 values, which means that while the overlap might be moderated, the boundary can deviate significantly in some regions.

Looking at the details, the Aorta generally achieves a decent dice score in most cases, meaning the model captures the main portion of the aorta well, while its HD95 can be moderate, signifying a few boundary discrepancies but not extreme. On the other hand, the gallbladder has a very low dice score in nearly all cases, indicating the model is missing or misidentifying the gallbladder, and also the near-zero HD95 often corresponds to no detection at all, rather than a perfect boundary match.

In addition, the Left and Right Kidneys have the dice score ranging from moderate to low across cases, suggesting partial organ coverage. About the HD95 values, these classes can spike in certain cases, implying large boundary misalignment. The right kidney in particular often shows higher HD95. Liver is among the highest dice scores across multiple cases, reflecting relatively robust segmentation. However, the HD95 can occasionally be substantial, highlighting boundary outliers or shape irregularities. The pancreas consistently exhibits lower dice scores, highlighting the challenge of segmenting this organ due to its small size and variable shape. Additionally, its HD95 values remain relatively high, indicating frequent boundary misalignments and difficulties in precisely delineating its contours.

In contrast, the spleen achieves moderate-to-high dice scores across most cases, reflecting a more reliable segmentation performance. Its HD95 values are typically lower than those of many other classes, suggesting that the model captures its boundaries with greater accuracy and consistency. The stomach falls within a mid-range for dice scores, with noticeable variations between cases. Its HD95 values also fluctuate, likely due to its deformable nature and differences in shape and size across patients, making boundary alignment more challenging in certain instances.

TABLE I
MEAN DICE AND MEAN HD95 PER CLASS

Class	**Mean Dice**	**Mean HD95**
Aorta	0.7649	27.3662
Gallbladder	0.0000	0.0000
Left Kidney	0.6721	40.2664
Right Kidney	0.6656	57.6495
Liver	0.8832	48.1648
Pancreas	0.2332	41.7625
Spleen	0.7396	23.2705
Stomach	0.5641	26.3754

The given table I provides information about the mean dice score and mean HD95 by organ classes in the pretrained TransuNet model.

Overall, it can be seen that the liver class has high Dice but still has a sizable HD95, indicating strong coverage but with certain boundary outliers. Otherwise, the gallbladder class was completely missed due to small size or insufficient examples.

In particular, the model achieved a mean Dice score of 0.7649 for the aorta, indicating a relatively strong overlap between the predicted segmentation and the ground truth. However, the corresponding Mean HD95 of 27.3662 suggests that although the overall segmentation is fairly accurate, the model exhibits some boundary deviations. In stark contrast, the gallbladder class obtained a mean Dice score of 0.0000 with a Mean HD95 of 0.0000, which indicates that the model completely failed to detect this structure. Such a result may be attributed to the inherent difficulty of segmenting small and variably shaped organs.

Furthermore, the left kidney and right kidney yielded mean Dice scores of 0.6721 and 0.6656, respectively, with Mean HD95 values of 40.2664 and 57.6495. These figures suggest that while the model manages to capture a substantial portion of the kidney regions, there remains a significant discrepancy along their boundaries, particularly for the right kidney. Conversely, the liver demonstrates a high level of segmentation accuracy, with a mean Dice score of 0.8832, although its Mean HD95 of 48.1648 indicates that some boundary inaccuracies persist. Additionally, the pancreas presents a challenge for the model, reflected in a low mean Dice score of 0.2332 and a Mean HD95 of 41.7625, while the spleen and stomach achieve moderate performance with mean Dice scores of 0.7396 and 0.5641, and Mean HD95 values of 23.2705 and 26.3754, respectively.

VI. CONCLUSION

Transformers are known as architectures with strong innate self-attention mechanisms. In this report, we presented all mechanisms, architectures of the TransUNet model, did some experiments and analysed the final results. To fully leverage the power of Transformers, TransUNet was proposed, which not only encodes strong global context by treating the image features as sequences but also well utilizes the low-level CNN features via a U-shaped hybrid architectural design.

ACKNOWLEDGMENT

This work was supported by a PhD. Vo Hoai Viet at the University of Science, Ho Chi Minh City. Our team hopes that the professor will review this proposal report and give us comments and satisfactory solutions for this topic.

REFERENCES

[1] "Synapse multi-organ segmentation dataset," Available at https://www.synapse.org/Synapse:syn3193805/wiki/217789, 2024.

[2] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," 2016. [Online]. Available: https://arxiv.org/abs/1605.06211

[3] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," 2015. [Online]. Available: https://arxiv.org/abs/1511.08458

[4] J. Chen, Y. Lu, Q. Yu, X. Luo, E. Adeli, Y. Wang, L. Lu, A. L. Yuille, and Y. Zhou, "Transunet: Transformers make strong encoders for medical image segmentation," 2021. [Online]. Available: https://arxiv.org/abs/2102.04306

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762