Final Project Report for CS 184A/284A, Fall 2019

Project Title: "UW-Madison GI Tract Image Segmentation — Track healthy organs in medical scans to improve cancer treatment" From Kaggle Competition
Project Number(name): Unlimited Team

Student Name(s)
Yiru Tang, 18958047, yirut2@uci.edu
Rong Mu, 58682019, rmu1@uci.edu

1. Introduction and Problem Statement

In cancer treatments, radiology oncologists adjust the direction of X-ray in order to increase the dose towards tumors and avoid stomach and intestines. This is a time-consuming and labor intensive process to segment the position of stomach and intestines from the tumor. This project will use deep learning techniques to automatically segment tumors from the important organs by training neural net models on MRI scans images. The result will be evaluated with the dice coefficients and 3D Hausdorff distance for observing how accurate the segmentation boundary is.

2. Related Work

The inputs we obtain from the dataset are MRI images that are represented by features of each pixel. To adaptively learn spatial hierarchies of features, convolutional based neural networks that perform pixel-wise classification can be a reasonable choice to start with. So far, many models have been proposed for this purpose. One foundation model is called fully convolutional neural network (FCN)[1], which only performs convolution (and subsampling or upsampling) operations. One example of FCN is SegNet[2] that uses pooling indices from encoders to non-linear upsampling of corresponding decoders. Another model proposed by a German machine learning researcher, Ronneberger, is U-Net[3]. Instead of using pooling indices, it uses the entire feature maps from encoder to concatenate with decoder matrices to perform convolution. The U-Net is one of the most popular models for biomedical segmentation problems, so we choose it to be the major model for this project. Furthermore, we might look into some more complex and advanced models such as U-Net++[4] might reduce the semantic gap between the encoder and the decoder.

There are two potential technical approaches that we mainly consider. First, we will implement a U-Net as a baseline architecture, and try to improve the model performance by substituting the basic FCN encoder with other types of encoders, e.g. ResNet and EfficientNet. The reason why we choose U-Net as our first experimenting model is that our dataset contains a heavy amount of images but not all of them have the training annotations. With a relatively small number of effective data, U-Net can segment images accurately, leading to a potentially high performance. Also, U-Net can extract features at each level of the encoder when conducting downsampling, which are combined with the features in the decoder, so that the model is trained on more features during upsampling and will get better

[1] [1411.4038] Fully Convolutional Networks for Semantic Segmentation
[2] https://arxiv.org/abs/1511.00561
[3] https://arxiv.org/abs/1505.04597
[4] https://arxiv.org/abs/1807.10165

performance. Then, we will try to implement the improved version of U-Net, U-Net++, as the baseline architecture with different encoders. If we successfully implement it, we will get to compare the performance of those two model architectures.

3. Data Sets [at least 1 page]

In this project, we plan to use the GI tract scans image dataset provided by this Kaggle competition. The image dataset contains 38,496 MRI scans in total for patients in multiple days. The given dataset can be divided into two parts — a **train** folder containing MRI images, and a **train.csv** file indicating corresponding labels for each training image. Images in the train data folder are classified by patients, and subdivided by scans per day. They are stored as png files and varied in shapes (360x310, 234x234, 276x276), so that we perform data transformation before the training process. One example of MRI scans is provided below (figure 1).

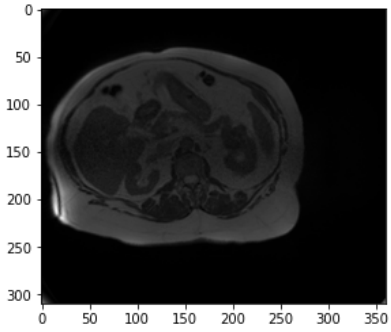


Figure 1. a slice image with a shape of 360 x 310

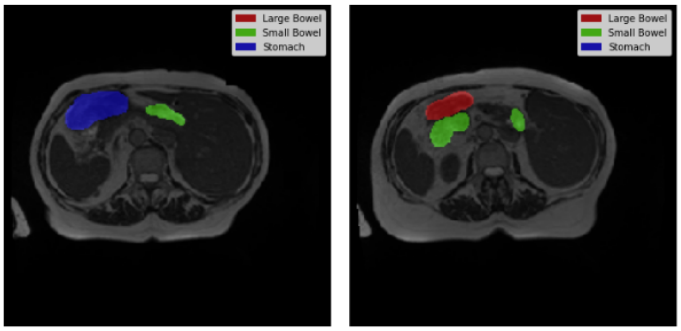


Figure 2. Overlayed masks for segmentation of 3 organs in grayscale in the scans folder

The target/label images are provided as segmentation arrays called Run Length Encode Mask(RLE). Instead of outputting a mask image, this method encodes the location of foreground objects in segmentation by recording the start pixels and how many pixels after each of those starts in a list. In addition, there are three classes of labels associated with each scan image. They are organ parts of the large bowel, small bowel and stomach respectively. We combine three classes of segmentation to one image, with different colors, so that when we feed labels to our model, our model can be trained to not only segmented GI tracts as all, but also three classes of those organs separately.

Not all images in the train dataset have corresponding segmentation labels, and for each MRI scan, it can be associated with only one or two of the classes, as shown in the figure 2. In the below figure, we compare the number of images that are either with some classes of segmentation (empty=false), or with no segmentation annotated at all (empty =true). The segmented training images

are slightly more than no label images.

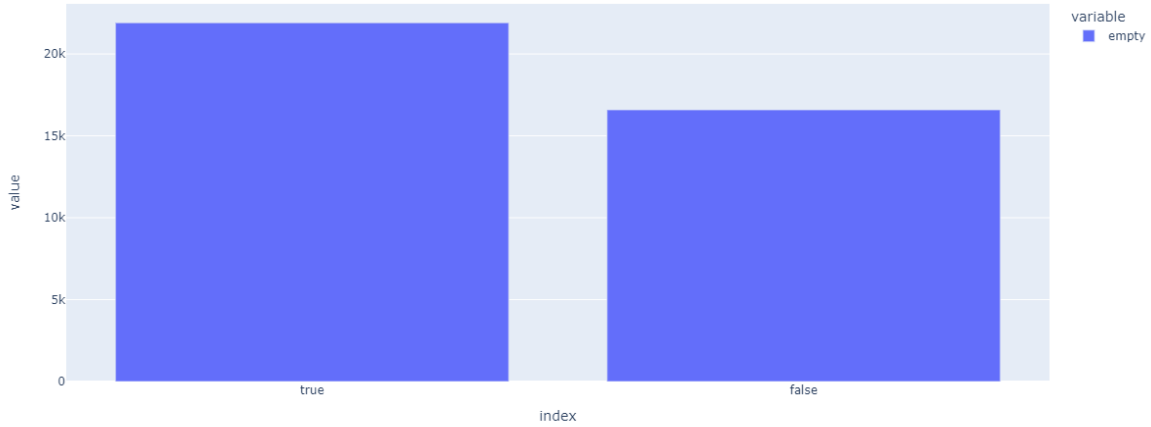


Figure 3

4. Description of Technical Approach [at least 1 page]

4.1 Data cleaning and Preprocessing

There are two options for us to preprocess the data and use it to train the model — with a grayscale mask of all three classes of segmentation combining together, or with one RGB mask that annotates each class in different colors. The first choice only involves a single dimension of the target layer, which makes this project a semantic segmentation problem. However, this process loses certain information and simplifies the problem we are addressing. Thus, we then tried out the other way to represent label images, which stacks three gray-scale labels together and forms a three dimension target. An illustration of the process can be seen in the pipeline below (figure 4).

As we introduced in the above section, the dataset is diverse in length and width, so the first step is to cut the image data to the same size, which is 224x224. To ensure that cutting the edges won't affect aligning with labels, and since the labels are also generated by our decoder, we choose to firstly generate all labels with the same size of its corresponding MRI scans, and then use a transformation package to slice them equally. In addition to the slicing, the external transformation package also provides us tools to rotate, rescale and translate the GI tract in the picture. Adding this variation helps

our model to better identify segmentation areas and avoid overfitting.

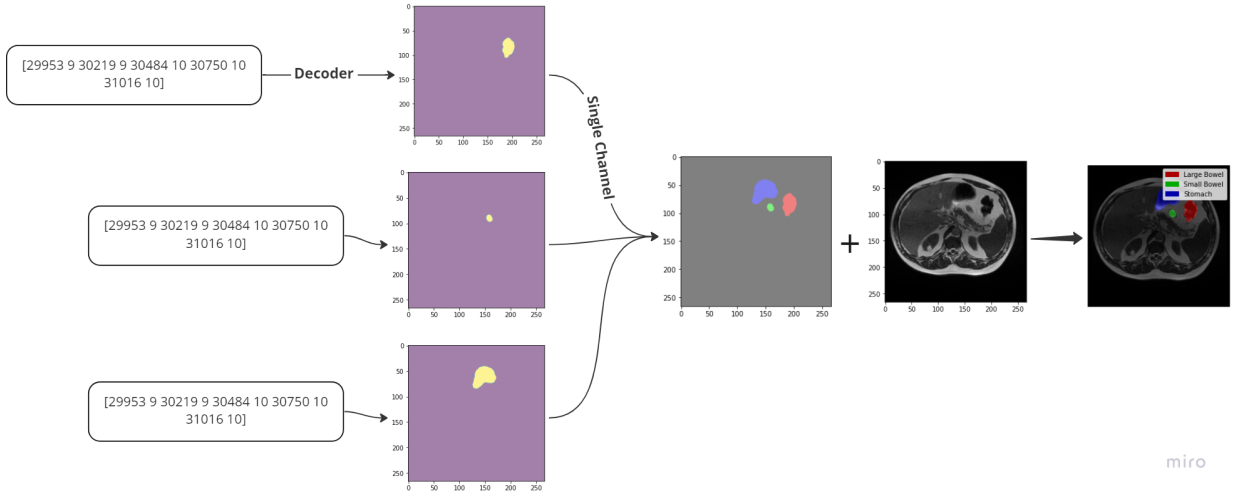


Figure 4

4.2 Model selection

We use the external library of [segmentation model built on pytorch] to construct our U-net model. It is using an encoder called efficientnet-b1, and an encoder depth of five. In-channel is three because we are using the RGB format of image data, and it should be matched to our 3D label images. The decoder channel is (256, 128, 64, 32, 16) by default. The class is defined to be three representing the large bowel, small bowel and stomach part of the segmentation. After dividing the data to training, validation, and test dataset, we define a batch size of 20 for the getter function in the data loader to perform. Finally, since this is a multilabel classification problem, we choose the softmax to be the activation function.

4. Software [at least ½ a page]

We have mainly built two critical parts of this project. The preprocessing takes a huge chunk of this project in respect of the variated training data and encoded target images. We decode RLE arrays to grayscale/RGB labels and then transform the image to numpy arrays, and we update a dataframe to combine all information like classes' labels, image paths, patient's id in one dataframe for the data loader to obtain training data. The other key part is that we reference a Kaggle Notebook[5] which provides a way to build dataset and dataloader and a library to run UNet. Based on those approaches we referenced, we are trying to define our own encoder to embed into the U-net model. This part will be discussed in detail in the next section.

The whole project is constructed in the sequence of data generating/preprocessing, build data loaders while transforming data, model construction, train and cross validation, and lastly, prediction and report. Below are some important libraries or packages that we used during the whole process.

[5] Reference: https://www.kaggle.com/code/awsaf49/uwmgi-unet-train-pytorch

Table 1: External Library/Package

Functionality	Tools
Image Transformation	albumentations
Model Structure	Segmentation_model.pytorch
Image Plotting Tool	colorama
Tracking and Visualization Tool	Weight and Bias
Cross Validation	sklearn

5. Experiments and Evaluation [at least 1 page, preferably 2 or 3]
[This is a critical part of your final report and the section where I expect to see the most new material compared to your progress report]
Describe in detail both (a) how you set up your experiments, including what metrics and methods you used for evaluation (test sets, cross-validation, user studies, etc), and (b) what results you obtained (ideally in the form of tables, graphs, etc), e.g., comparing the accuracies different methods and baselines. In this section you can add to your "basic results" by reporting additional comparative results on sensitivity of your approach to different algorithmic choices, e.g., how did performance depend on vocabulary size? On document length? On whether you removed stop-words or not? Did including parts of speech help? And so on.

Before we choose stratified K fold cross validation methods to split our training, validation, and test datasets. There are both labeled and unlabeled images with different proportions in the original dataset. Instead of using traditional cross validation methods, using this method guarantees for each fold the same proportion of labeled and unlabeled images are maintained, which makes each fold a better representation of the entire dataset. We split the whole dataset into five folds, and we keep fold 1-fold 4 as our training dataset, the entire fold 0 as our validation dataset, and the labeled images of fold 0 as our test dataset. Also, we use training dataset and validation dataset for training the model, optimize the parameters by the model automatically, use validation dataset to calculate the dice coefficient, and use test dataset to predict the segmentations.

Then, we move on to build and train the model. At the very beginning we view our problem as image segmentation without differentiating classes, so we process the labels using only one channel, which combines the masks of different classes together in a one-dimensional image. And we replicate a baseline UNet model using segmentation-model-pytorch library as mentioned above. The dice coefficient is about 0.6, and the prediction is empty. We think it is because the prediction is to segment the images into different classes, using one-dimensional images doesn't make sense.

So we decided to process the labels with differentiation of classes. But we defined the problem at this point as multi-label classification since there are overlaps between segmentation masks of different classes. After we processed the labels to three-dimensional images, for each of which is one of the

classes, we trained the same UNet model using images and the new version of labels. We set epoch=5 and learning rate=0.002 at the very beginning. It turns out that the training and validation loss decreases from 0.84 to 0.50, but the validation dice coefficient remains 0.2255, which is very low. But we observed that the learning rate is decreasing during the training. We tried to increase the number of epochs to 10 and 15 and decrease the learning rate to 0.001. But it remained the exact same accuracy.

Since it takes a long time to run the training process, we then choose to use the Wandb platform to train the model, for its advantages of tracking and visualizing the training and validation loss and memory usage as well as helping tune the hyperparameters. We set epoch=5 and epoch=15 with learning rate = 0.002 and 0.001. Here is the dice coefficient accuracy:

number	epoch	learning rate	dice coefficient	Note
(1)	5	0.001	0.6496	
(2)	15	0.001	0.7238	
(3)	15	0.002	0.8799	run in Kaggle Notebook

Run Summary, Training Loss, and Validation Loss

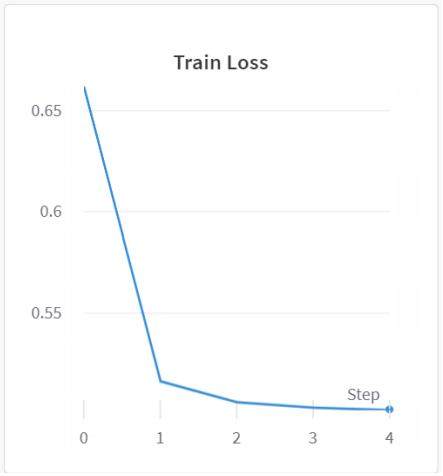
for the first process referred to number (1) in the above table

```
Training complete in 1h 12m 23s
Best Score: 0.6496
```
Waiting for W&B process to finish... **(success).**

Run history:

Run summary:

Best Dice	0.64963
Best Epoch	5
Best Jaccard	0.64963
LR	0.00098
Train Loss	0.50225
Valid Dice	0.64963
Valid Jaccard	0.64963
Valid Loss	0.50138



(1)

Run Summary, Training Loss, and Validation Loss

for the first process referred to number (3) in the above table

```
Training complete in 2h 42m 52s
Best Score: 0.8799
```

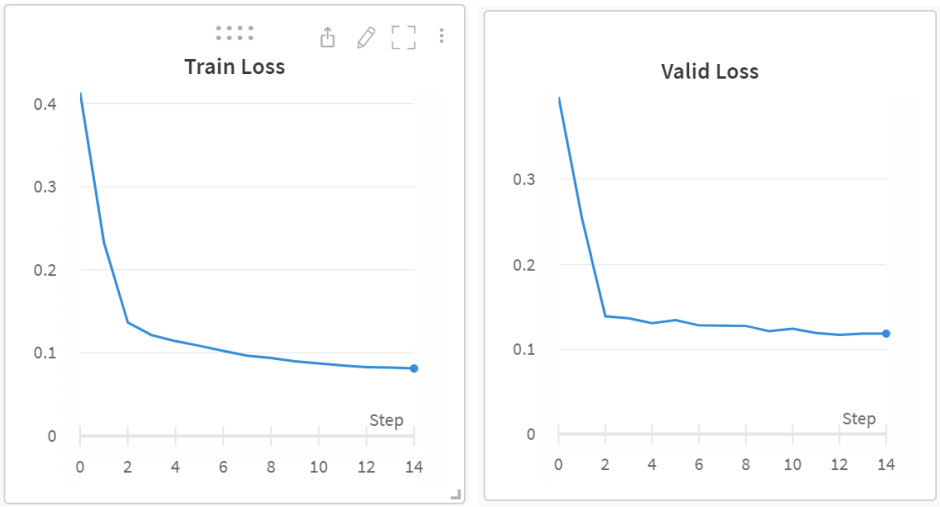
Waiting for W&B process to finish... **(success).**

Run history:

LR	▇▇▇▆▅▄▃▂▁
Train Loss	▇▂▁▁▁▁▁▁▁
Valid Dice	▁▆▇▇▇▇▇▇▇
Valid Jaccard	▁▆▇▇▇▇▇▇▇
Valid Loss	▇▂▁▁▁▁▁▁▁

Run summary:

Best Dice	0.90785
Best Epoch	15
Best Jaccard	0.87989
LR	0.0
Train Loss	0.08122
Valid Dice	0.90785
Valid Jaccard	0.87989
Valid Loss	0.11858



(3)

It turns out that the best performance is to use 15 epochs and 0.002 learning rate to train the model. We do realize that we need to try more values for the hyperparameters, we will do this in the future work.

We also tried to use the VGG 16 model as well, but the training loss is already 0.0001, so we thought it might be overfitting there, and we will also find it out later.

6. Discussion and Conclusion [at least ½ a page]

Because of the complexity of the project context, we have gained a lot of experience managing a huge amount of data. We learnt more advanced tools to transform data, and we also learnt to use external tools like scheduler, GPU, and performance tracking platform to help us develop this project. Surprisingly, the tools we use had a huge impact on what the final result is, especially in terms of utilizing the tracking platform wandb, and we figured out that it has performed a great hyperparameter tuning for us. Also, we have found efficient tools for dividing dataset and performing cross validation.

In the process, we spent a lot of time trying to understand the input data and we failed again and again testing different tools impact on our performance. We are still implementing an additional encoder and hopefully we will use it in the future.

7. A separate page on *Individual Contributions*

Name**:** Yiru Tang

In this project, I have experimented with multiple external libraries and have done model selection, model construction, parameter tuning, and image visualization. Rong and I collaborated on the task of data preprocessing, model training, presentation preparation, and report writing.

Name: Rong Mu

In this project, I have created labels of training dataset, done data preprocessing and visualization, created a demo Jupyter notebook for visualization. I have also implemented an extra encoder to be embedded into U-net. I collaborated with Yiru Tang for class presentation preparation, report writing and model tuning, testing and evaluation process.