110-2 Project – Liver tumor segmentation with abdominal CT scan

醫工二 鍾孟辰

b09508011@ntu.edu.tw

National Taiwan University

Taipei, Taiwan

1 Abstract

In this report, I'll present the progress of my project. For my project, I aim to use U-net to segment the liver into the tumor and the normal parts with abdominal CT scans. This semester, I've built a U-net network and finished preprocessing. Moreover, I've also done some research for future work, such as different loss functions/ accuracy for segmentation, augmentations, transfer learning, and more.

2 Introduction

Liver cancer is the second most common cancer in Taiwan. In addition, the death rate of liver cancer in Taiwan is much higher than in other countries. What's worse, the treatments are usually having more limits in Taiwan due to the high prevalence rate of hepatitis. Even if cured, 70% of patients would have a relapse within five years. Therefore, it is important that the doctors could find the tumor earlier. The earlier they found the tumor, the higher survival rates and better treatment a patient can have. This is where the CAD system come in handy. With the help of deep learning, the doctors could easily find the indiscoverable tumor.

In this report, I'll not show a well-trained model because I have some difficulties in programming. However, I'll show how I'm going to deal with the images and how I'll train the model. In the end, I'll describe my future work.

I use two datasets on Kaggle in this project, "Liver Tumor Segmentation" and "Liver Tumor Segmentation-part 2". Both of the datasets contain abdominal CT scan images, and 130 images in

total. In this report, I'll show the methods, including neural network, preprocessing, and training process, in section 3. Some results of preprocessing and a neural network are shown in section 4. Following is a discussion and conclusion, where I'll show some ideas for future work and reflections on this semester.

3 Methods

3.1 Neural network

While deep neural networks are becoming increasingly powerful tools for solving medical imaging segmentation problems, we still need large datasets for model training. However, high quality-labelled datasets are usually expensive and rare. Moreover, there are other limits, such as hardware. Therefore, the dimension reduction technique is applied to reduce computation, which in CNN or here, U-Net, is called encoding. Then, dimension increase takes place in the second half of the model to make predictions, which we call the second half "decoder".

Among other networks, a deep neural network called U-Net usually performed more successfully. The main reason that makes U-Net a better model is its connecting path between the decoder and encoder. During encoding, some features or noises would be eliminated in order to reduce the dimension and make sure the model ignores the noises. Nevertheless, when our task is to detect the abnormal parts in medical images, such as Macular degeneration in eyes or tumors in our bodies, these abnormal parts are usually erased during encoding. In U-Net, with those connecting paths between encoder and decoder, we can better preserve those lesions. As a result, I would like to implement U-Net in this project.

The structure of the U-Net is shown in figure 1. As its name, the structure of U-Net looks like the alphabet U. The left side is the encoder, in charging with dimension reduction, and the right side with red squares is the decoder, which is responsible for dimension increase. Last, the yellow squares show the connecting layer between the encoder and decoder, which are the parts that make U-Net more successful. In the encoder, max pooling is used as down sampling to decrease the dimension and a sequential convolution layer are aim to produce a more precise output. Between each convolution layer is a ReLu activation function. In the decoder, 2×2 convolution is used to increase the dimension and the rest is the same as in the encoder. Last, the output is the same size as the input images and with the segmented map.

I built a U-Net network exactly like Figure 1. The code can be found in Section 7.

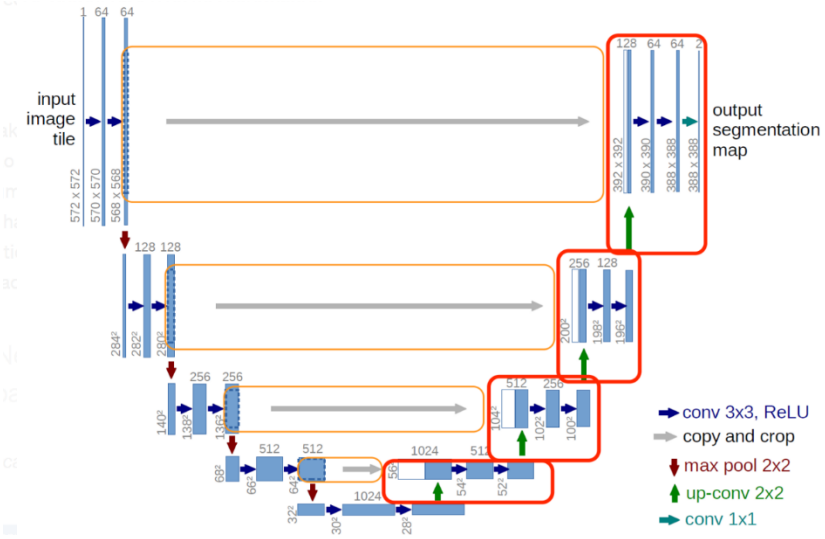


Figure 1. Structure of U-Net

3.2 Training process

During the training process, loss function and accuracy are essential. In this section, I'll show some loss and accuracy functions for segmentation.

Loss function:

There are numerous loss functions for segmentation, and each of them focuses on different characteristics of datasets. Here, I would like to introduce three loss functions, which are binary cross-entropy (or so-called pixel-wise cross entropy loss), focal loss, and last, dice loss (soft dice loss specifically).

- Binary Cross-Entropy (pixel-wise cross entropy):

  As the name suggests, pixel-wise cross entropy examines each pixel individually and repeats over all pixels and averages. The function is shown as:

$$Loss = -\frac{\sum_{classes}\{y_{true} \times \log(y_{pred})\}}{number\ of\ pixels}$$

  The function is intuitive. We compare the differences between the predictions and the ground truths of each pixel and then average. However, it is also easy to find a situation in which the loss function would fail. That is, when the target parts are small, the loss function would fail. For example, in Figure 2, only a few pixels have been labeled. Let's say, the size of the image is $96 \times 96\ pixels$ and only 10 pixels are

labeled, which is $^{10}/_{96 \times 96} = 0.1\%$ of the image. Under this circumstance, if the model gives a prediction as Figure 3, the loss would be extremely low as 0.1%. As a

Figure 2. The ground truth for the example.

Figure 3. The prediction for the example.

result, we can say that the loss function failed. Yet, in equal data distribution among class scenarios, binary cross-entropy works well.

- Focal Loss:

Although Focal loss can be seen as a variation of Binary Cross-Entropy, unlike Binary Cross-Entropy, Focal loss can focus on high-imbalanced datasets depending on the set of parameters. Focal loss aims to down-weighting the outliers (hard examples) and tries to ignore those easy samples. To give a more specific concept, in the paper "Focal Loss for Dense Object Detection", they mention the ratio of background and foreground can reach 1:1000, which is about 0.1%. As the result, the example made previously might find a solution with Focal loss. Before we look at the definition of Focal loss, let's take a recall on CE (cross-entropy):

$$CE(p, y) = \begin{cases} -\log(p), & if\ y = 1 \\ -\log(1-p), & otherwise \end{cases}$$

To make a convenient notion of Focal loss, we abbreviate the definition as follows:

$$CE(p, y) = CE(p_t) = -\log(p_t), p_t = \begin{cases} p, & if\ y = 1 \\ 1-p, & otherwise \end{cases}$$

Let's consider a loss function defined as follows:

$$FL(p_t) = -(1-p_t)^{\gamma} \log(p_t)$$

From Figure 4 we can see that the CE function gives a high loss value with easy examples($p_t < 0.5$), while a low loss value is given with easy examples($p_t > 0.5$).

Though the loss of a hard sample is higher, the accumulation of 1000 easy examples is still bigger than a hard sample loss. As consequence, the model would learn more about the background. The author proposes a method for addressing the class imbalance, which is to introduce a weighting factor $\alpha \in [0,1]$ for class 1 and 1-α for class -1. The function was defined as α-balanced CE loss and can be written as:

$$CE(p_t) = -\alpha \log(p_t)$$

Yet, the function was merely scaling the loss values. The problem that with enough example loss accumulated, the loss of hard examples might be ignored. To solve the problem, the author adds the modulation factor: $(1-p_t)^{\gamma}, where\ \gamma \geq 0$. Now we can finally define the Focal loss as:

$$FL(p_t) = -(1-p_t)^{\gamma} \log(p_t)$$

Finally, the author mentioned that by adding weight to the function, the accuracy increased slightly. Therefore, the final version of the Focal loss function is defined as follows:

$$FL(p_t) = -\alpha(1-p_t)^{\gamma} \log(p_t)$$

Last, look at Figure 4, we can see that with a higher γ, the loss function has a better concentration on hard examples. It is also notable that Cross-Entropy is actually the special case of Focal loss function. When $\alpha = 1$ and $\gamma = 0$, the function became

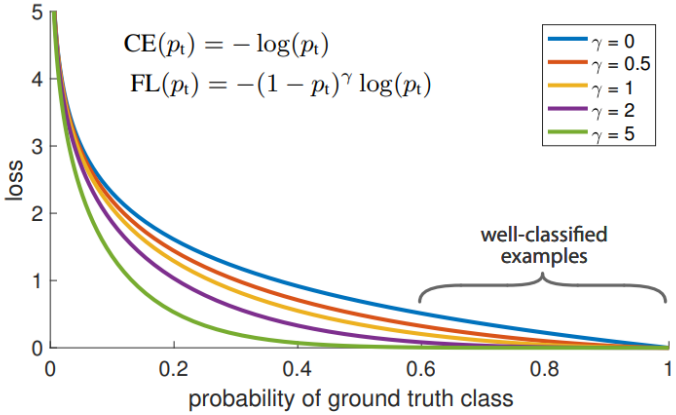


Figure 1. We propose a novel loss we term the *Focal Loss* that adds a factor $(1 - p_t)^{\gamma}$ to the standard cross entropy criterion. Setting $\gamma > 0$ reduces the relative loss for well-classified examples ($p_t > .5$), putting more focus on hard, misclassified examples. As our experiments will demonstrate, the proposed focal loss enables training highly accurate dense object detectors in the presence of vast numbers of easy background examples.

Figure 4. Figure from the paper "Focal Loss for Dense Object Detection"

exactly like CE (blue line in Figure 4).

- Dice Loss (soft Dice Loss):

Dice loss, especially soft Dice loss, is widely used to evaluate image segmentation loss. In order to formulate a loss function that can be minimized, we simply define Dice loss as $1 - Dice\ coeficient$. This function is known as soft Dice loss. The difference between Dice loss and soft dice loss is that soft Dice loss can be calculated with probability and without a threshold.

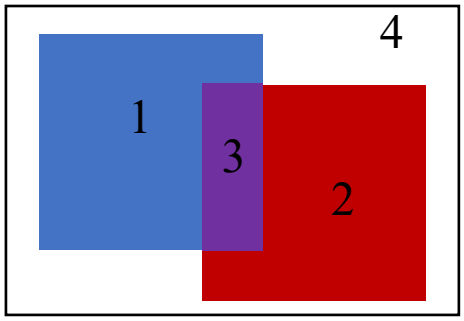


Figure 5. Example for Dice Coefficient

Dice Coefficient:

Look at Figure 5, suppose the red square is the prediction while the blue square is the ground truth. The union part (purple) can be considered correct predictions. Now let's give those areas some numbers, 1 is the blue part, 2 is the red part, 3 is the purple, and 4 is the rest.

Now, the dice coefficient is defined as follows:

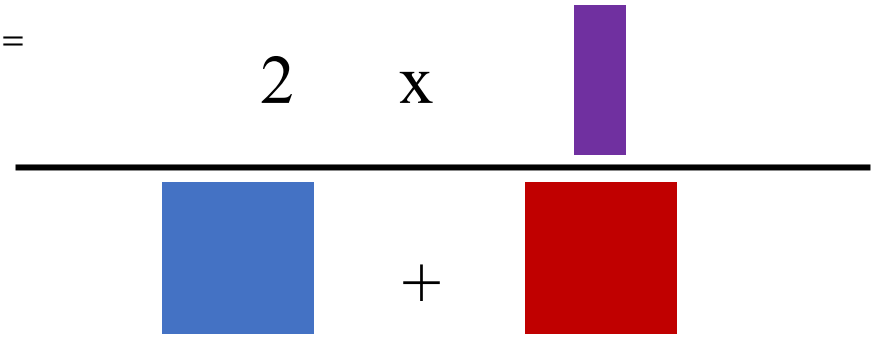
$$Dice = \frac{2 \times \sum_{pixels} y_{true} y_{pred}}{\sum_{pixcels} y_{true}^2 + \sum_{pixels} y_{pred}^2}$$

Also written as:

$$Dice = \frac{2 \times area\ 3}{(area\ 3 + area\ 1) + (area\ 3 + area\ 2)}$$

To be more visualize:

$Dice =$



With the concept of dice coefficient and soft dice loss, let's consider the condition here:

1. Total number of pixels for predicted segmentation and ground truth annotation: 200
2. Target: Area of Overlap = 0%
3. Background: Area of Overlap = 95%

$$Dice\ loss = 1 - dice\ coefficient = 1 - {}^{0+95}/_{200} = 52.5$$

With the result, we can see that even though two classes (target and background) are imbalanced in the image, the soft Dice coefficient still functions well.

Accuracy:

The accuracy can be defined as $1 - loss\ function$ and the pros and cons are the same as their loss function. Therefore, I'll not introduce them again. Instead, I would like to raise three questions about accuracy:

1. How would I know if the accuracy is high enough to be considered a well-trained model?

   We could compare our performance to other works' or the benchmark of the dataset to evaluate whether the model is well-trained.
2. Can the accuracy/loss function be different in the training process and testing process?

   As long as we could give an explanation, the accuracy could be different. The most common explanation is that function A works better during the training process. However, we could evaluate our performance more scrutable with function B.
3. How would I know if the model is ready for clinical or commercial usage?

   We could compare the dice coefficient of manual labeled data and model prediction. If the model performs close to manual labeled or even better, it is good enough for clinical usage. In fact, with this standard, we might not always need a high-accuracy model. For instance, if a disease is hard to label, the dice coefficient of the manual label is low or the diversity of two doctors is high, then a relatively better result is enough for clinical usage.

In this project, I'll use Dice coefficient and soft Dice loss. Yet, after other parameters are set, I'll try different loss function to see if the accuracy increases.

3.3 Preprocessing

In preprocessing, I plan to adopt mini-batch and standardize the dataset. However, for now, I've only finished the labeling part and the result is shown in Section 4.

3.4 Augmentation

Compared with conventional machine learning, deep learning-based models perform better only under big datasets. Unfortunately, usually, the small dataset is more readily accessible due to the expensive cost of labeling. When deep learning-based models were trained with small datasets, overfitting is easily observed. An easy solution to the problem is augmentation, in which we create more diversity in the image by randomly flipping them, adding noises, or randomly rotating the images.

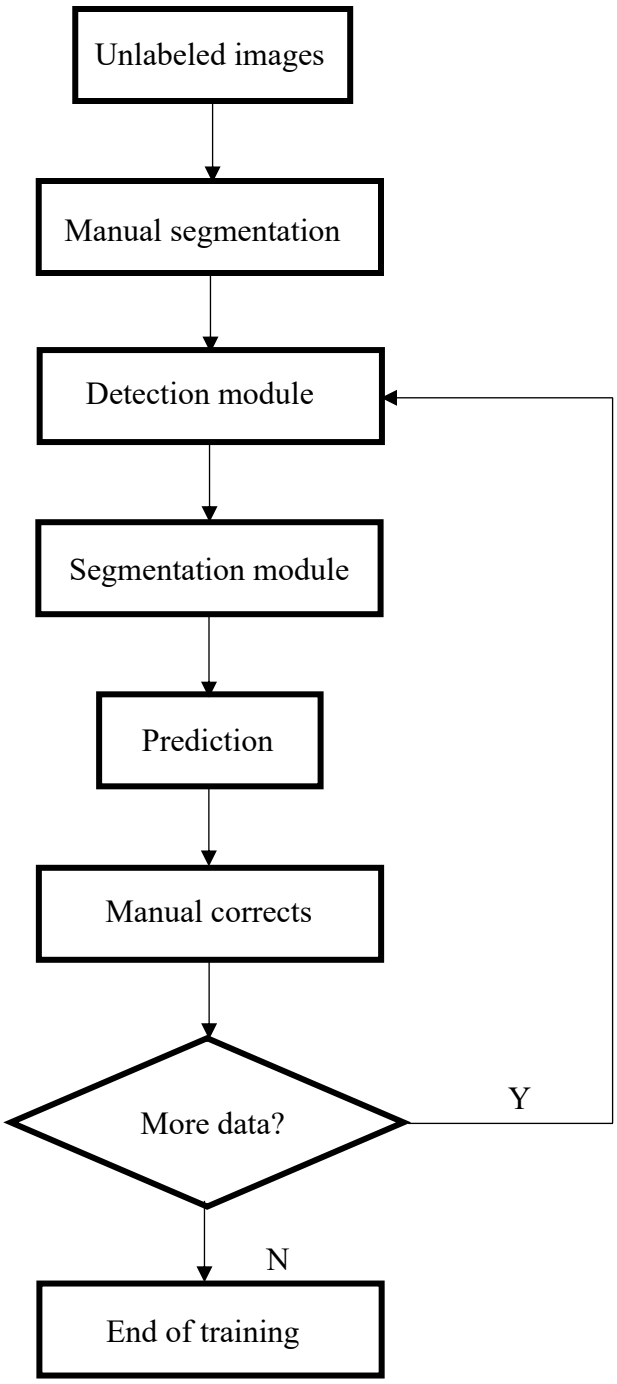
Unlike other methods I'm going to introduce in the next subsection, augmentation is easy to program. In addition, many researchers have devoted themselves to different-based augmentation strategies. In other words, there are plenty of options to choose from and to experience. In another course, I used a Perlin noise-based augmentation strategy to enhance my performance. I also experienced what impact would different noises cause. In that project, I made the conclusion that only the noises which can present the pattern could make the accuracy higher. Otherwise, they might even lower the accuracy. The variant between different noises is pretty big and unsteady. In addition, there are limits to augmentation's efficiency. Therefore, other approaches are essential for further increasing the accuracy.

3.5 Approaches on improving performances

In this subsection, I would like to show the concept of transfer learning and active learning and their characteristic.

- Active learning:

  Active learning is usually used in segmentation and could reduce the manual label time. The flow diagram of active learning is shown as follows:



Let's consider we have 50 unlabeled images for training data. First, we split it into 20, 20, and 10 images. Then, labeled the first 20 images manually and use them to train a model. Next, use the trained model to do the segmentation on the next 20 images. The predicted images were then undergoing a manual correction and added to the training dataset. Now, we have 40 labeled images in our training dataset. Still, we have 10 more

unlabeled images. After training the module again with a bigger dataset, the rest of the 10 images were served as the validation data and then manually corrected before adding into the training dataset. When there are no more images, the training process ended. In this way, we do not need to label 50 images manually. Instead, we only need to label 20 of them, and the rest are labeled by the model and some corrections.

Besides shortening the labeling time and cost, active learning was also found to improve the network by iterative learning. This method is easy and efficient. However, it could not be performed without experts correcting those predictions. As a result, it is not suitable to adopt in this project.

- Transfer learning:

There are four situations for different kinds of transfer learning, shown in Table 1.

		Source Data (not directly related to the task)	
		labelled	unlabeled
Target Data	labelled	Fine-tuning Multitask Learning	Self-taught learning Rajat Raina , Alexis Battle , Honglak Lee , Benjamin Packer , Andrew Y. Ng, Self-taught learning: transfer learning from unlabeled data, ICML, 2007
	unlabeled	Domain-adversarial training Zero-shot learning	Self-taught Clustering Wenyuan Dai, Qiang Yang, Gui-Rong Xue, Yong Yu, "Self-taught clustering", ICML 2008

Table 1. 李宏毅教授投影片

In this project, I'll focus on Fine-tuning.

Fine-tuning is common in the deep learning training process. With a pre-training model and fine-tuning with some parameters, the performances in the target source usually converge quicker and better. However, if the target dataset is small, overfitting might happen. Therefore, two major ways to transfer the pre-training model are introduced as follows:

1. Conservation Training:

   In conservation training, we need to build a new model first. Then, adding some regularization to constrict the new model to predict the same result as the pre-training model under the same input. With the restriction, we could make sure the fine-tuned model still performs well on source data.

2. Layer Transfer:

   In layer transfer, we copy parameters in some layers. Usually, in the classification task, we take parameters to form the front layers. It is because, in image classification, the first few layers are usually responsible for feature extraction. Those features are basic, such as some lines and corners of the image. As we transfer those layers to the new model, the model would not need to learn that basic information again.

There are also some limits to adopting a pre-training model. That is when adopting the pre-training model did not make the performance better nor shorten the training duration.

Since I adopt U-Net in this project, which is a common network, and my hardware resource is very limited, I consider fine-tuning a good option for me.

4 Results

4.1 Neural network

A summary of parameters in U-Net can be found in the file “Summary of U-Net parameters.txt” on the URL in section 7.

4.2 Preprocessing

The image1 is the original image and image2 is a sample image after labeling. Image3 shows the mask information in vector.

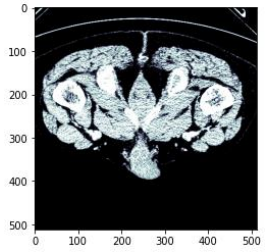


Image 1. Original CT scan

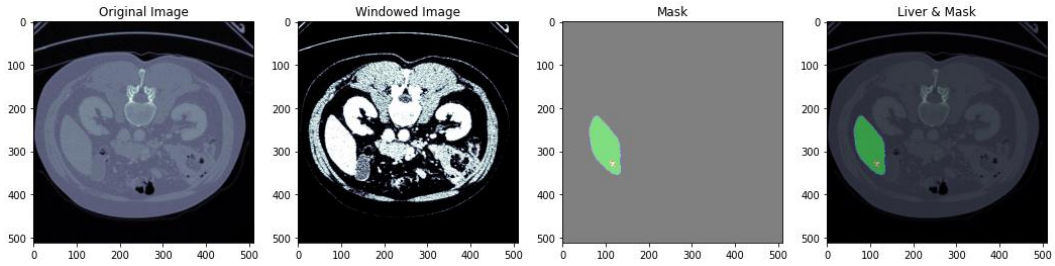


Image 2. After preprocessing and labeling

```
(512, 512)
[[     0 255677]
 [     1   6360]
 [     2    107]]
<matplotlib.image.AxesImage at 0x7f2154446a50>
```

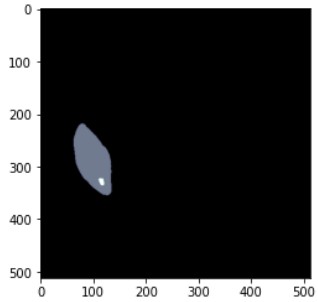


Image 3. The mask

5 Discussion

In my future work, the first thing to do is to give some results. Then, I'll try to apply different methods, such as transfer learning, adding different noises, or modifying some of the layers. A series of experiments will be performed.

6 Conclusion

Although unable to give a result, I've learned many. From how to choose and build a network to methods that might improve the performance, such as applying augmentation data and transfer learning. In addition, I've also learned some skills in preprocessing, and I'll take classes next semester to learn more. Besides deep learning, I've also learned a lot about programming and experience. Those are valuable foundations for making a good project. Last, I planned to continue this project during the summer vacation. I hope by the end of the summer vacation, I'll be able to run out some results and give some improvement.

7 Programing

The code is available in the following URL:

https://github.com/CMC2002/110-2-Liver_Tumor_Segmentation.git

8 Reference

Datasets:

Liver Tumor Segmentation: https://www.kaggle.com/datasets/andrewmvd/liver-tumor-segmentation

Liver Tumor Segmentation - Part 2: https://www.kaggle.com/datasets/andrewmvd/liver-tumor-segmentation-part-2

Medical image analysis: https://www.kaggle.com/code/dmisky/medical-image-analysis

Preprocessing of Liver Tumor Segmentation part 1 and part 2: https://www.kaggle.com/code/mohanmkshirsagar/liver-tumor-segmentation-part-1-and-part-2

Neural Networks and Deep Learning: http://neuralnetworksanddeeplearning.com/

3Blue 1Brown Series: Deep learning: https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

Designing Your Neural Networks: https://towardsdatascience.com/designing-your-neural-networks-a5e4617027ed

Transfer learning: https://medium.com/%E6%88%91%E5%B0%B1%E5%95%8F%E4%B8%80%E5%8F%A5-%E6%80%8E%E9%BA%BC%E5%AF%AB/transfer-learning-%E8%BD%89%E7%A7%BB%E5%AD%B8%E7%BF%92-4538e6e2ffe4

Pre-training: https://medium.com/%E6%88%91%E5%B0%B1%E5%95%8F%E4%B8%80%E5%8F%A5-%E6%80%8E%E9%BA%BC%E5%AF%AB/transfer-learning-%E8%BD%89%E7%A7%BB%E5%AD%B8%E7%BF%92-4538e6e2ffe4

Loss functions: https://dev.to/_aadidev/3-common-loss-functions-for-image-segmentation-545o

How to Evaluate Image Segmentation Models: https://towardsdatascience.com/how-accurate-is-image-segmentation-dd448f896388

Metrics to Evaluate your Semantic Segmentation Model: https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2

U-Net: https://amaarora.github.io/2020/09/13/unet.html#understanding-input-and-output-shapes-in-u-net

Deep Learning for Image Segmentation: U-Net Architecture: https://heartbeat.comet.ml/deep-learning-for-image-segmentation-u-net-architecture-ff17f6e4c1cf

Shruti Jadon: A survey of loss functions for semantic segmentation: https://www.researchgate.net/publication/342520628_A_survey_of_loss_functions_for_semantic_segmentation

Tsung-Yi Lin: Focal Loss for Dense Object Detection: https://arxiv.org/abs/1708.02002

Focal Loss: https://chih-sheng-huang821.medium.com/%E6%A9%9F%E5%99%A8-%E6%B7%B1%E5%BA%A6%E5%AD%B8%E7%BF%92-%E6%90%8D%E5%A4%B1%E5%87%BD%E6%95%B8-loss-function-huber-loss%E5%92%8C-focal-loss-bb757494f85e

Designing Your Neural Networks: https://towardsdatascience.com/designing-your-neural-networks-a5e4617027ed

Pre-training: https://medium.com/ai-blog-tw/deep-learning-%E4%BD%BF%E7%94%A8pre-training%E7%9A%84%E6%96%B9%E6%B3%95%E8%88%87%E6%99%82%E6%A9%9F-b0ef14e777e9

Transfer learning: https://medium.com/%E6%88%91%E5%B0%B1%E5%95%8F%E4%B8%80%E5%8F%A5-%E6%80%8E%E9%BA%BC%E5%AF%AB/transfer-learning-%E8%BD%89%E7%A7%BB%E5%AD%B8%E7%BF%92-4538e6e2ffe4

Taehun Kim Active learning for accuracy enhancement of semantic segmentation with CNN-corrected label curations: Evaluation on kidney segmentation in abdominal CT: https://www.nature.com/articles/s41598-019-57242-9

Hyun-Jin Bae A Perlin Noise-Based Augmentation Strategy for Deep Learning with Small Data Samples of HRCT Images: https://www.nature.com/articles/s41598-018-36047-2