Digital Image Processing (2023)
Final Project Report
Water Segmentation

Group: No.9
Authors:
311511022 邱政岡 311591023 燕新城 312581010 孟羽真

1 Introduction

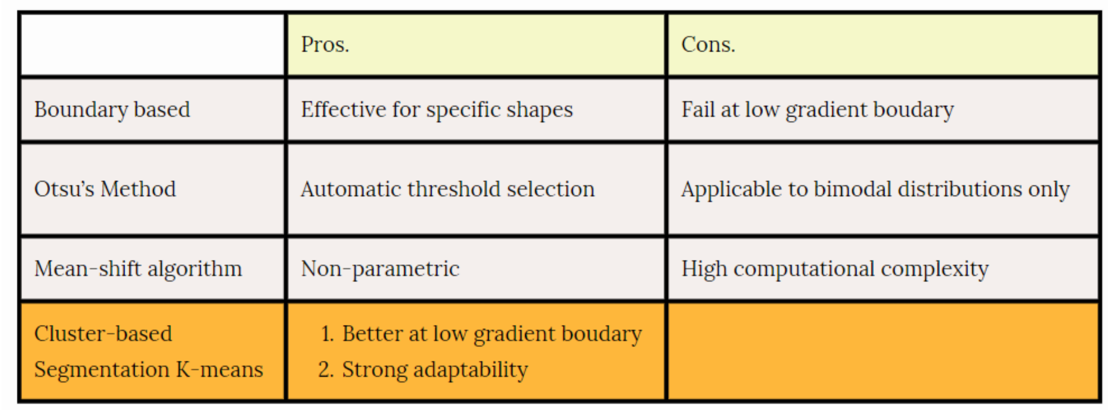
In segmentation area, benefited from flexibility and adaptability, DL method has been more and more popular today. But for a rather small dataset, said 60 images. What will the result be? We proposed a twice K-means method and use it to compare with the deep learning model. Analysis it's advantage and disadvantage.

This article will be organized as follow: Conventional method using the Cluster based method K-means with pre-processing and post-processing in chapter 2. Deep learning method using UNet++ architecture with dpn68b pre-trained backbone in chapter 3. Testing data results and discussion in chapter 4.

2 Conventional method

2.1 Method Choosing

We have research four methods as well as their Pros. and Cons. They all have their plus in some aspects. But as for water segmentation, we need to deal with low gradient boundary and all kinds of data distribution in diverse scenario. Regarding all the concerns, K-means is the best candidate in this problem.



	Pros.	Cons.
Boundary based	Effective for specific shapes	Fail at low gradient boudary
Otsu's Method	Automatic threshold selection	Applicable to bimodal distributions only
Mean-shift algorithm	Non-parametric	High computational complexity
Cluster-based Segmentation K-means	1. Better at low gradient boudary 2. Strong adaptability	

Table 1 Method Choosing

2.2 Process Flow

In order to simplify our K-means method, we need pre-processing and post-processing. In pre-processing, we increase the uniformity of light and color. In post-processing, we select and merge the water region.

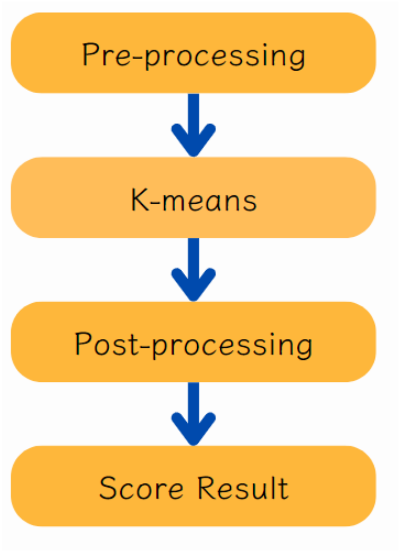


Figure 1 Process Flow

2.3 Pre-processing

2.3.1 Histogram equalization

We analyze the histogram and balance it to emphasize the shadow area.

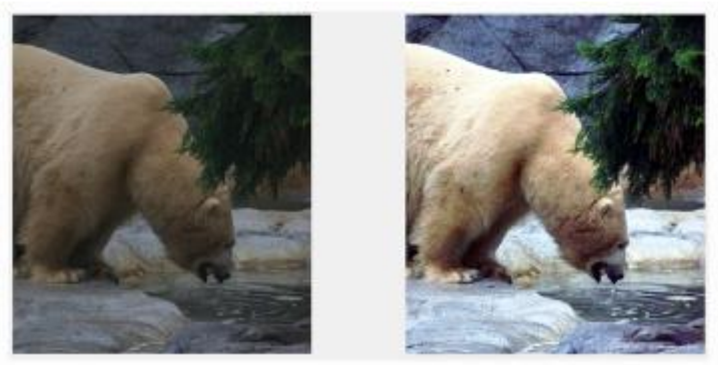


Figure 2 Histogram equalization

2.3.2 Color temperature balance

For where warm color is the main part, we balance it by the ratio of max RGB and current RGB.

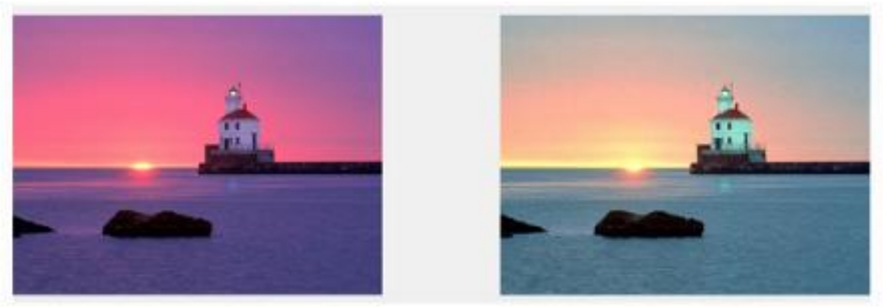


Figure 3 Color temperature balance

2.3.3 Local Laplacian filtering

We use local Laplacian filter to eliminate the details without damages to the edge.

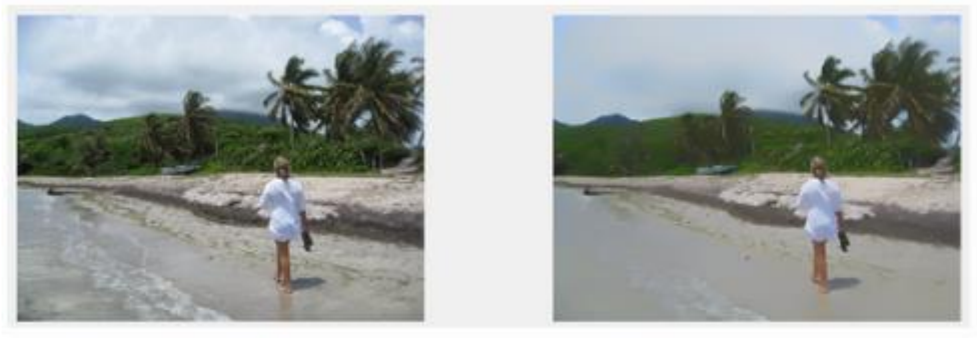


Figure 4 Local Laplacian filtering

2.4 K-means

We use K-means twice. The main difference between these two steps is the stop criteria. The first K-means stop when all the points had been visited. On the other side, Second time of K-means we stop by specify a fix cluster number. By combining together, we create as more clusters as we can at first one for precision and merge than in second one for post-processing.

2.4.1 First K-means for details

The algorithm includes Global classified, Iteration for new means, Merge, and repeat this three steps until go through all the points. At Global Classified, we first randomly choose a red point that hasn't been visited before. Then we compare all the points that haven't been visited with this red point and decide whether they are same cluster by the criteria of distance and color.

After we created this red area, we recalculate the points in it and get new means. And then we compare these new means with other exist clusters and determine whether they are same cluster and whether to merge then.

Repeat this iteration until all the points are calculated.

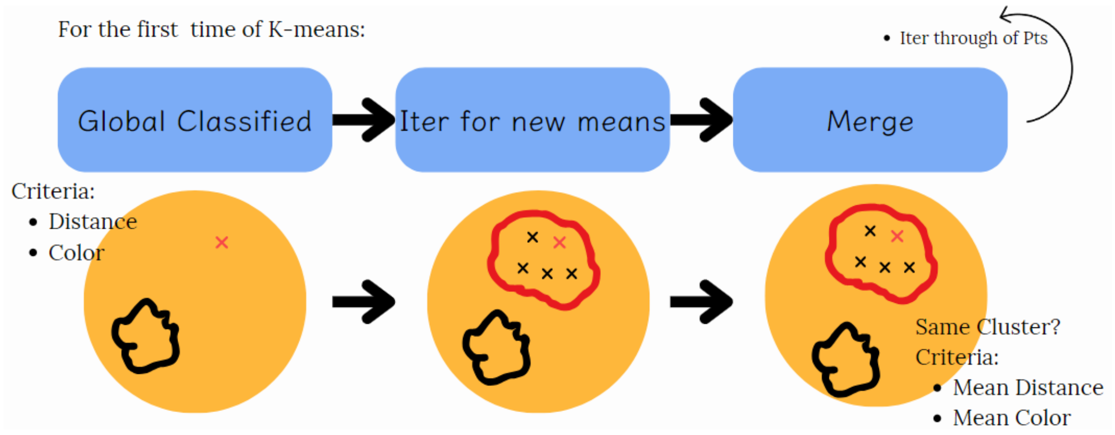


Figure 5 First K-means

Furthermore, computing time can be significantly reduced by resizing the image to half. Processing the dataset contains 60 images, computing time becomes 10 mins from 60 mins. At the same time, the accuracy does not impact or even better.

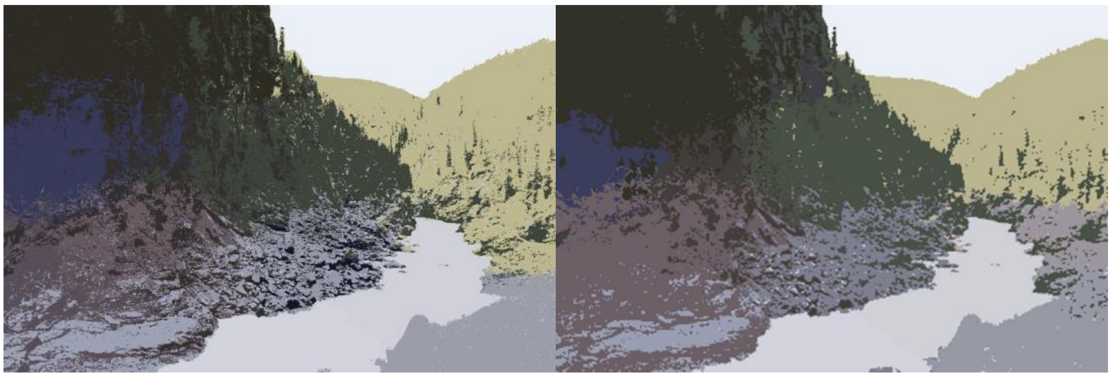


Figure 6 K-mean clustering result: (left) no resize (right) resize to half

2.4.2 Second K-means for merge

After first clustering, if the number of clusters is greater than 10. We apply a second K-means clustering by MATLAB image processing toolbox function to reduce the cluster number to 10.

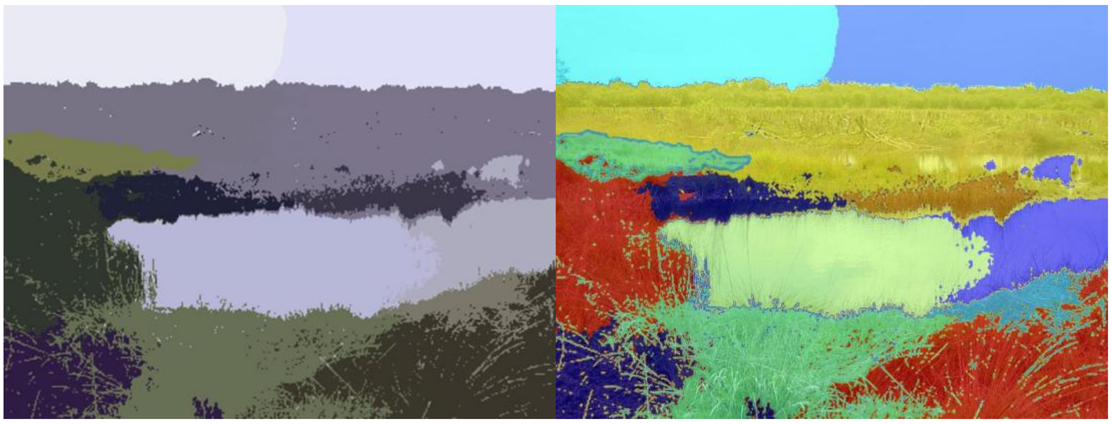


Figure 7 K-means clustering to 10 segments and labeling

2.5 Post-processing

2.5.1 Select and merge segments

After K-means clustering, the image is split into at most 10 segments. The water may contain one or more segments. We should manually select segments that are water.

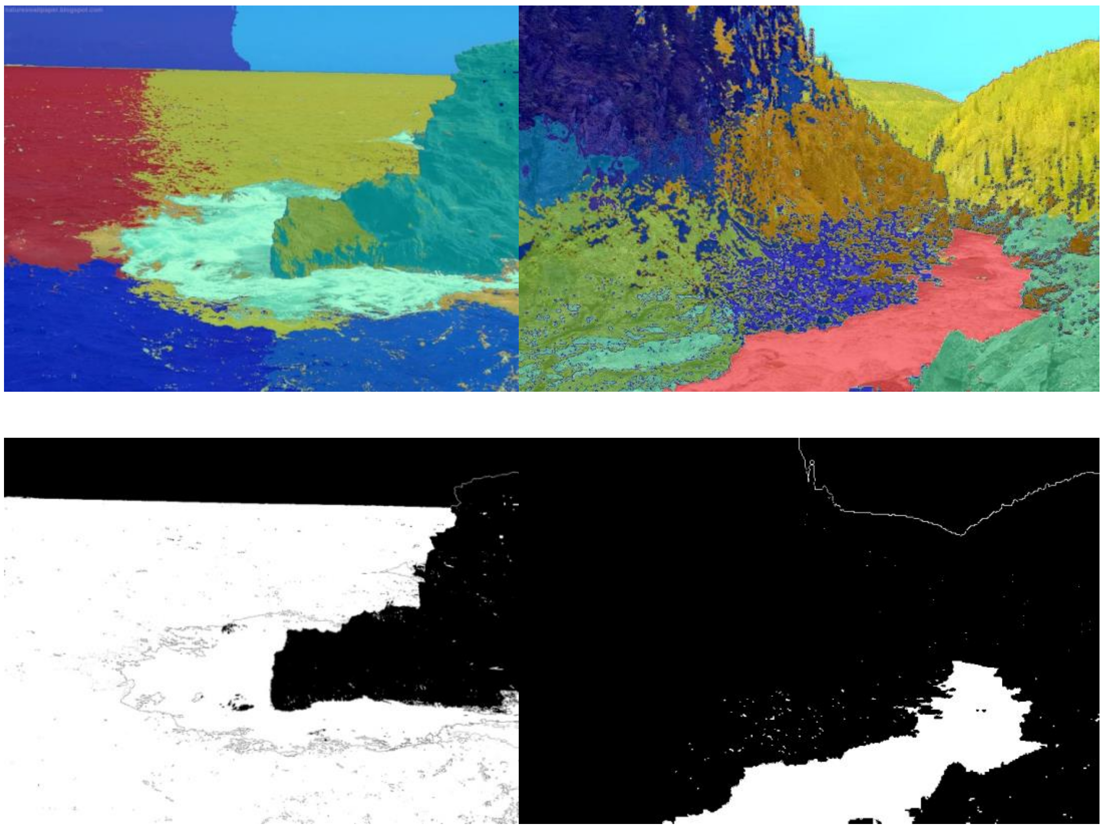


Figure 8 merge the selected segments and convert to binary mask image

(left up) water is split into multi segments (right up) water only one segment

(down) binary image of up image after merged

2.5.2 Closing of opening

Some small pieces are appeared because the complex feature or color difference.

The morphology image processing can eliminate these noise.

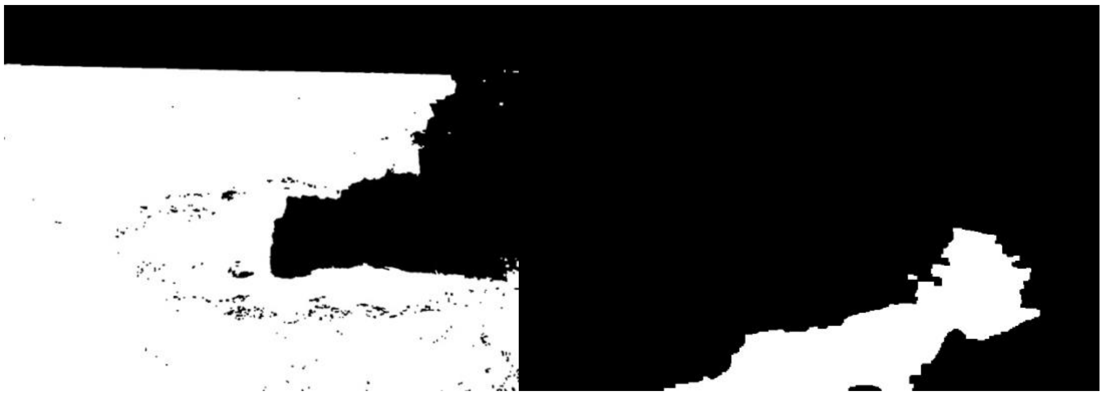


Figure 9 the closing of opening af the down image in figure3

2.6 Training data Result

In the Figure 10, we can see. Because color is an important feature of water, the water can be segmented by our processing.

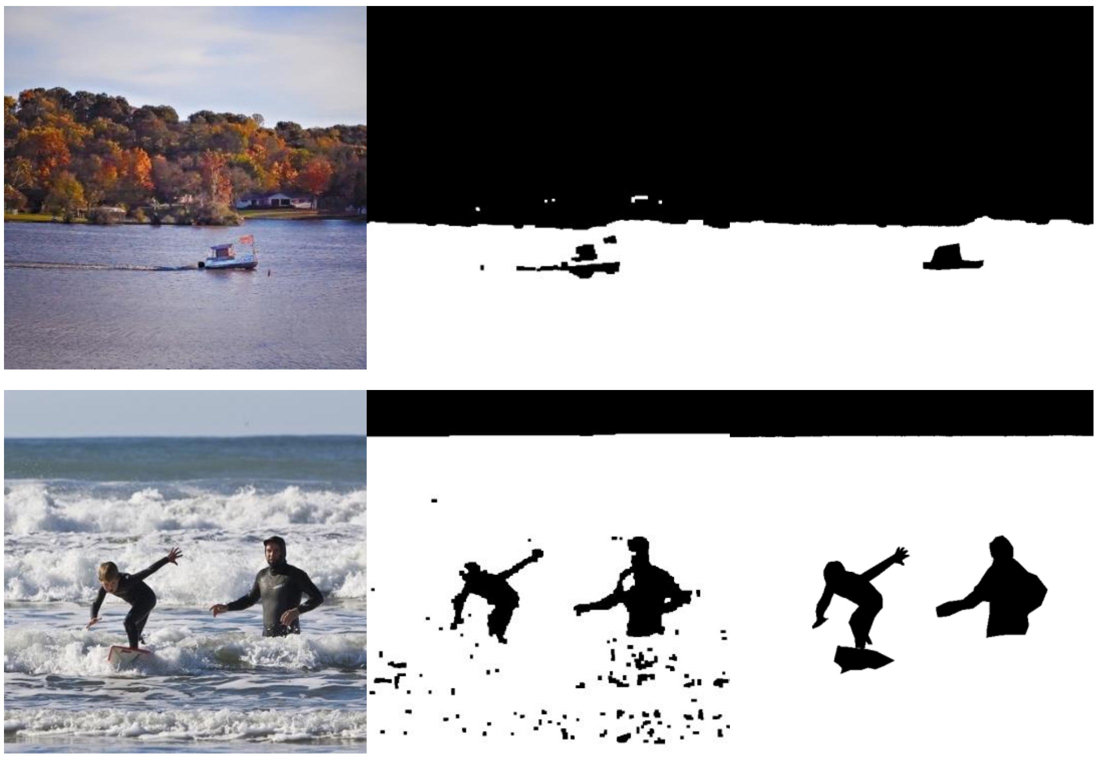


Figure 10 some good water segmentation by our method

(left) source image (middle) our result (right) ground truth

In Figure 11, some specific scenes that contain reflection or shadow are hard to segment.

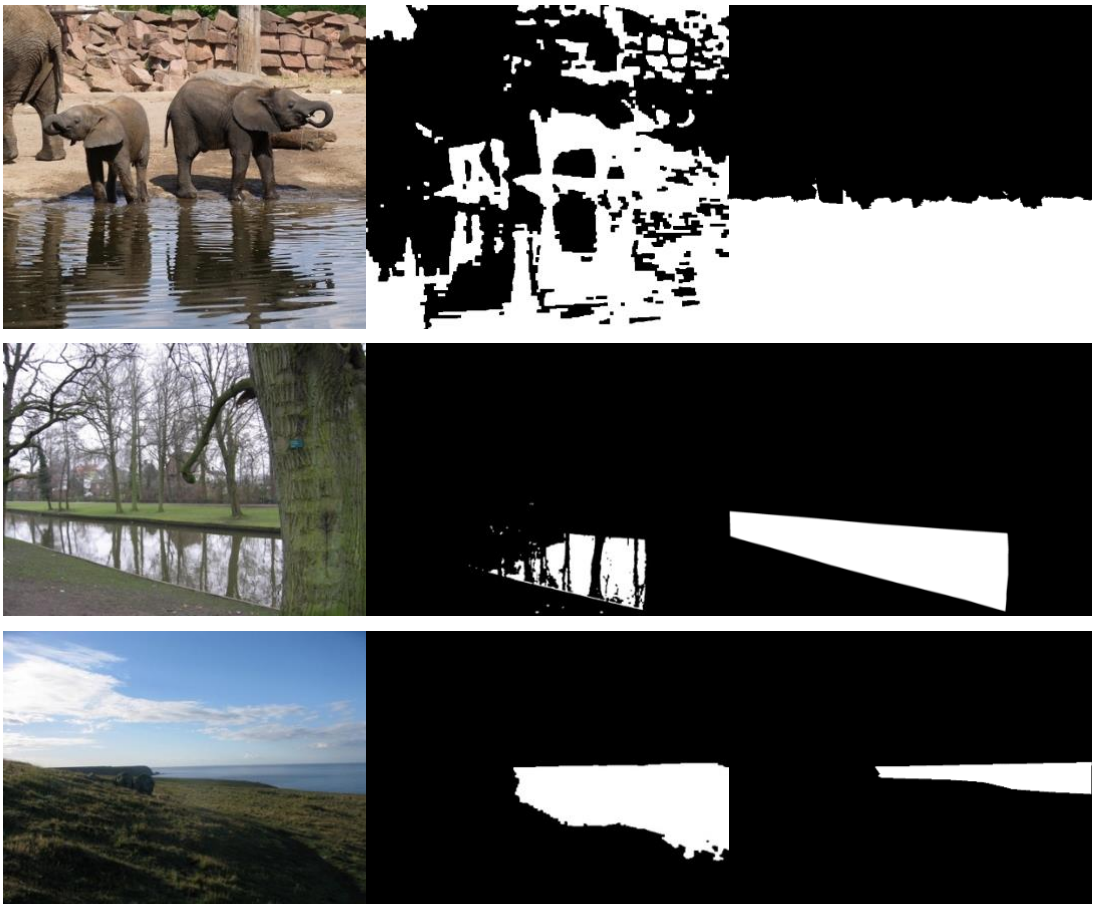


Figure11 some bad water segmentation by our method

(left) source image (middle) our result (right) ground truth

3 Deep learning-based method

3.1 Pretrained model + Data augmentation

訓練深度學習模型需要大量的 data，由於此次 training data 數量較少，導致要從頭開始訓練一個 image segmentation 的 DL model 非常困難，因此為了提高模型的 performance，採用 Pretrained Model 搭配 Data Augmentation。

3.1.1 Pre-trained Model

模型已在大量的影像資料上面做過別的任務的訓練(預訓練)，如 imagenet, imagenet+5k 等等，其預訓練後的模型本身具備一定的捕捉 feature 的能力，再透過 fine-tuning 方式套用到此次 water segmentation 的 task 上，即使在資料較少的情況下，預訓練模型也能實現良好的效能。

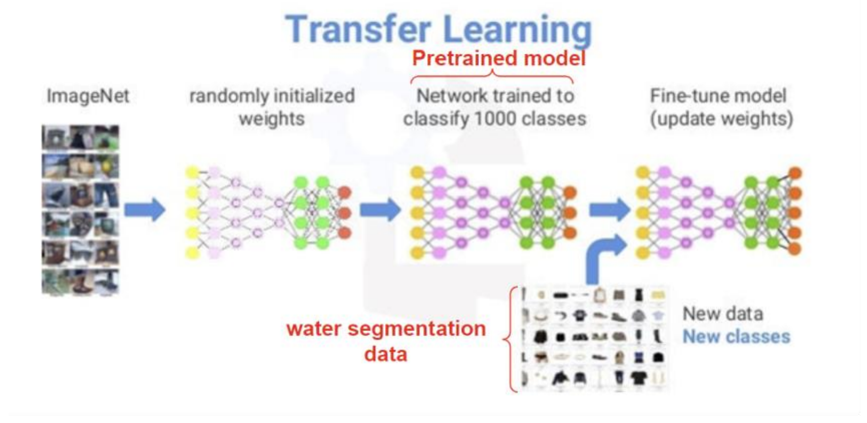

Transfer Learning
Pretrained model
ImageNet
randomly initialized weights
Network trained to classify 1000 classes
Fine-tune model (update weights)
water segmentation data
New data
New classes

Figure 12 *Transfer learning Flow [1]*

使用的 Pytorch framework 提供的 pretrained model [2]以及 pretrained dataset

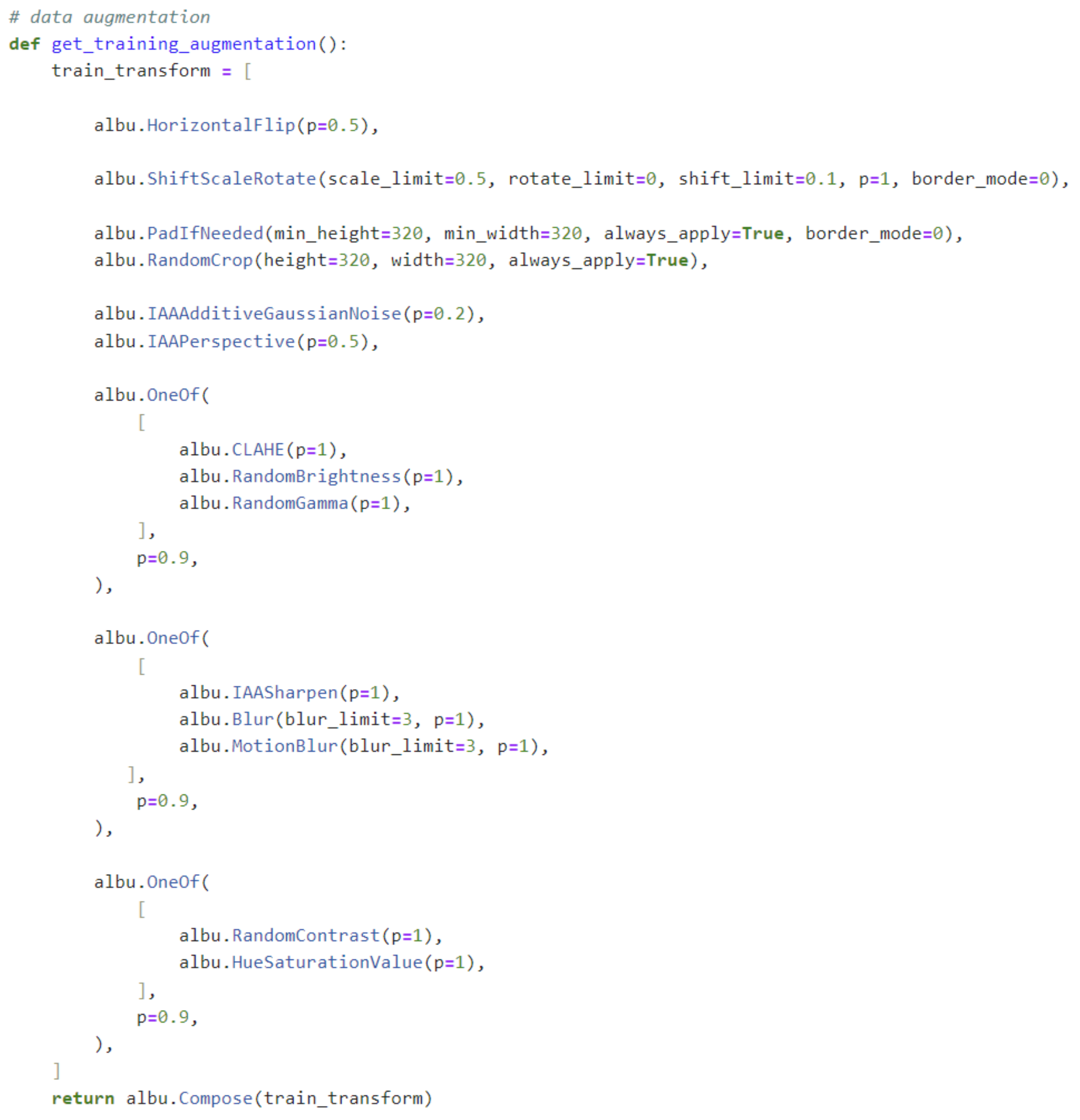
- resnet50 - imagenet
- resnet101 -imagenet
- dpn68b (Dual Path Networks) -imagenet+5k
- dpn92 (Dual Path Networks) -imagenet+5k
- efficientnet-b2 - imagenet
- efficientnet-b3 – imagenet

3.1.2 Data Augmentation

為了提高模型的效能，我們需要提供更多的 training data，以使模型能夠辨識更多的特徵，從而做出最終決定。一個有效解決資料不足的方法是資料增強（Data Augmentation），從現有數據中套用一些變化產生新的數據，進而增加可以用來訓練的資料量，它能夠有效提高模型的準確度並節省時間金錢。
使用 Pytorch 提供的 Albumentations library，套用速度快的同時，也支援多種 Augmentation 技術，適用於影像相關任務，下列為此次套用的技術

- HorizontalFlip：水平翻轉
- ShiftScaleRotate：移動（Shift）、縮放（Scale）和旋轉（Rotate）影像
- RandomCrop：隨機裁剪影像的一部分
- IAAAdditiveGaussianNoise：添加高斯雜訊
- IAAPerspective：改變影像視角
- CLAHE：在局部區域內，改善影像的對比度
- RandomBrightness：隨機改變影像的亮度值
- RandomGamma：隨機調整影像的 γ 值
- IAASharpen：讓影像銳利化，加強細節
- Blur：讓影像模糊化
- RandomContrast：隨機調整影像的對比度

◆ HueSaturationValue：隨機調整影像的色調（Hue）、飽和度（Saturation）和值（Value）



```python
# data augmentation
def get_training_augmentation():
    train_transform = [

        albu.HorizontalFlip(p=0.5),

        albu.ShiftScaleRotate(scale_limit=0.5, rotate_limit=0, shift_limit=0.1, p=1, border_mode=0),

        albu.PadIfNeeded(min_height=320, min_width=320, always_apply=True, border_mode=0),
        albu.RandomCrop(height=320, width=320, always_apply=True),

        albu.IAAAdditiveGaussianNoise(p=0.2),
        albu.IAAPerspective(p=0.5),

        albu.OneOf(
            [
                albu.CLAHE(p=1),
                albu.RandomBrightness(p=1),
                albu.RandomGamma(p=1),
            ],
            p=0.9,
        ),

        albu.OneOf(
            [
                albu.IAASharpen(p=1),
                albu.Blur(blur_limit=3, p=1),
                albu.MotionBlur(blur_limit=3, p=1),
           ],
            p=0.9,
        ),

        albu.OneOf(
            [
                albu.RandomContrast(p=1),
                albu.HueSaturationValue(p=1),
            ],
            p=0.9,
        ),
    ]
    return albu.Compose(train_transform)
```

Figure 13 Data Augmentation 程式碼片段

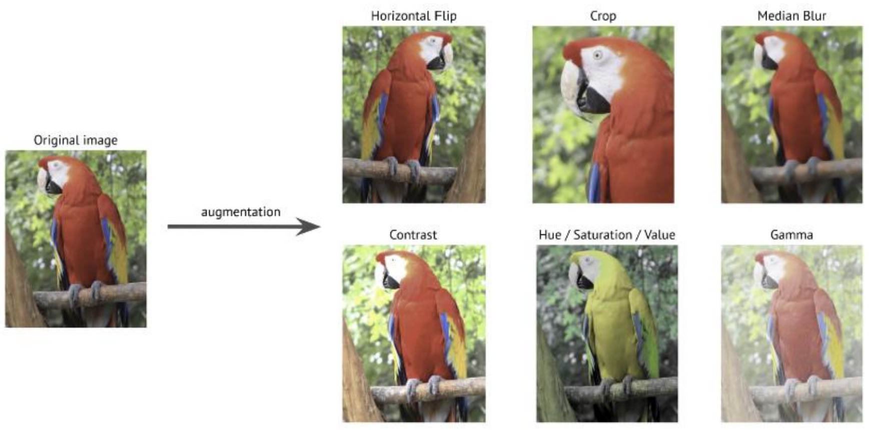


Figure 14 Data Augmentation 效果

優點

- 降低資料收集和資料標記的成本
- 透過賦予模型更多的多樣性和靈活性來改進模型泛化
- 提高模型在預測中的準確性，因為它使用更多資料來訓練模型
- 減少數據的過擬合

3.1.3 Experiment with data augmentation in combination with different pretrained models

總共 60 張照片，將其切分成 75% for training/17% for validation/8% for testing，下面實驗結果是採用相同的 data augmentation 方法，使用 UNet++、DeepLabV3+架構，搭配不同的 pretrained backbone model，在 testing image 上面得到的 segmentation 結果以及 average iou score。

pretrained backbone	resnet50	resnet101	dpn68b	dpn92	efficientnet-b2	efficientnet-b3
UNet++	0.47	0.396	0.64	0.57	0.64	0.54
DeepLabV3+	0.53	0.495	0.498	0.54	0.59	0.41

Figure 2 Testing IOU Score

從 Figure 11 [結果 1]可以看到此方法在 segmentation 效果還不錯，但在[結果 2]中，發現與水較相近顏色的區塊會造成模型混淆，容易也將其誤判成水的一部分。

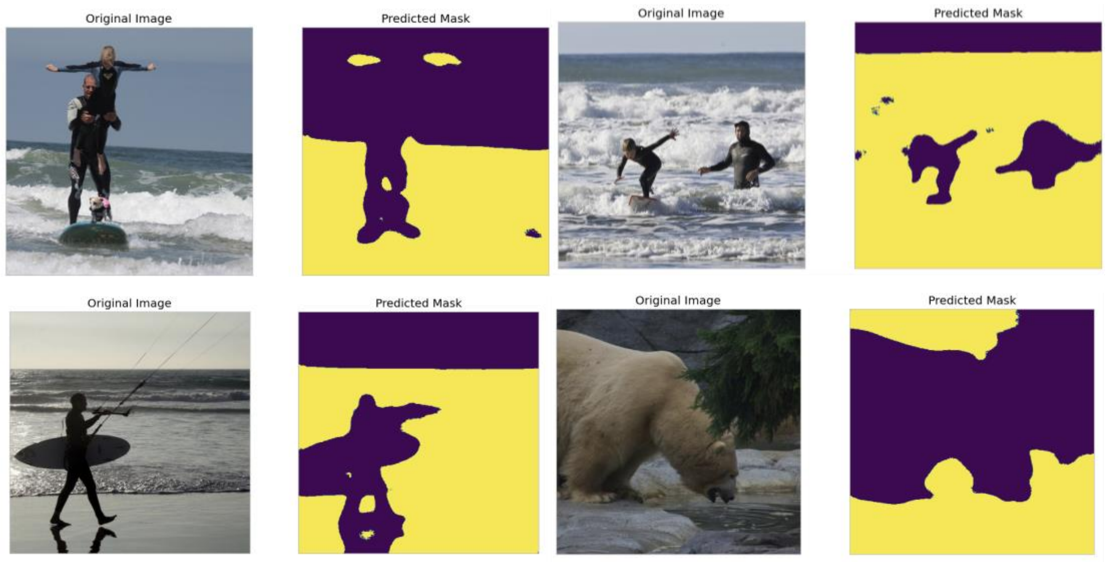


Figure 16 Testing result on UNet++ architecture with dpn68b pre-trained backbone

(左上、右上)結果 1 和 (左下、右下)結果 2

3.2 Ensemble learning + Data augmentation

Ensemble Learning 透過結合多個模型的預測結果來提高整體的預測效能，這種方法的核心思想是將多個模型的力量集合起來，以達到比單一模型更好的預測效果。此次採用類似 bagging 的方法，結合多個 fine-tuning 的 Pretrained model，再將這些 Pretrained model 的輸出做 average 來當作最後的 output，其概念如下圖所示。

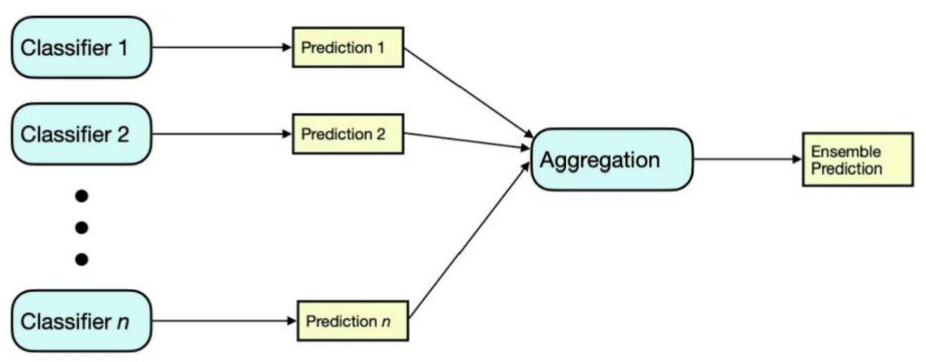
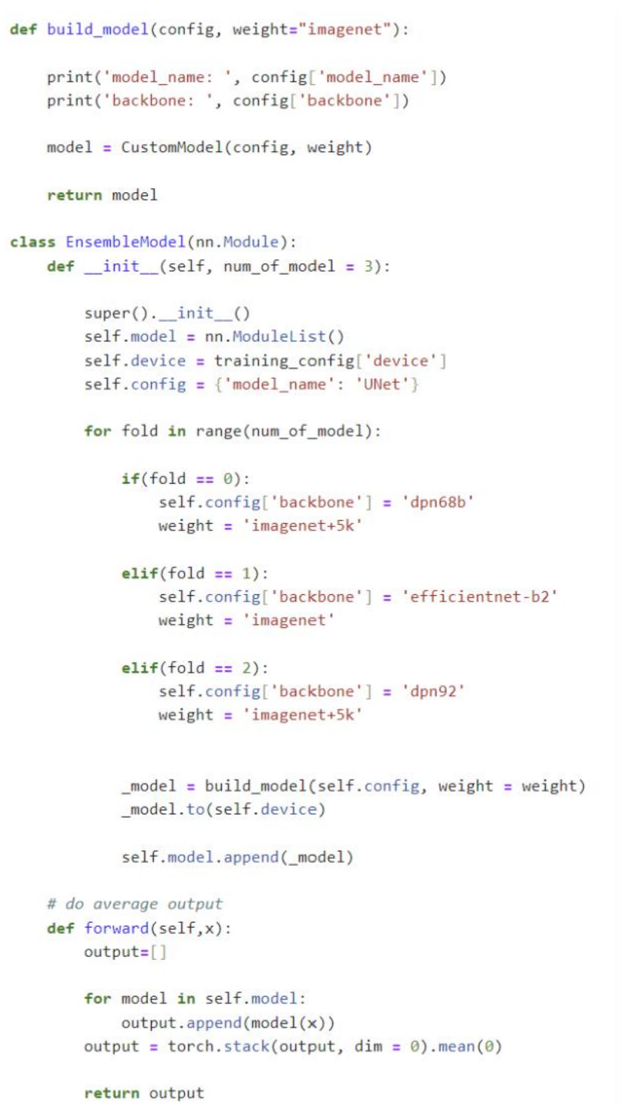


Figure 3 Ensemble learning 架構圖 [3]

3.2.1 Experimental results of Ensemble Learning combined with data augmentation

採用於上面 Testing average IOU score 分數最高的 3 個模型(UNet++ with dpn68b / dpn92 / efficientnet-b2 backbone)來做 ensemble learning。



```python
def build_model(config, weight="imagenet"):

    print('model_name: ', config['model_name'])
    print('backbone: ', config['backbone'])

    model = CustomModel(config, weight)

    return model

class EnsembleModel(nn.Module):
    def __init__(self, num_of_model = 3):

        super().__init__()
        self.model = nn.ModuleList()
        self.device = training_config['device']
        self.config = {'model_name': 'UNet'}

        for fold in range(num_of_model):

            if(fold == 0):
                self.config['backbone'] = 'dpn68b'
                weight = 'imagenet+5k'

            elif(fold == 1):
                self.config['backbone'] = 'efficientnet-b2'
                weight = 'imagenet'

            elif(fold == 2):
                self.config['backbone'] = 'dpn92'
                weight = 'imagenet+5k'


            _model = build_model(self.config, weight = weight)
            _model.to(self.device)

            self.model.append(_model)

    # do average output
    def forward(self,x):
        output=[]

        for model in self.model:
            output.append(model(x))
        output = torch.stack(output, dim = 0).mean(0)

        return output
```

Figure 4 Ensemble learning 程式碼片段

從下面 testingimage 中可以看出利用多個模型去做 segmentaion 效果會比使用單一 pretrain model fine-tuning 後來的好，外圍輪廓的地方分得更好的同時，也比較能辨認出球的部分。

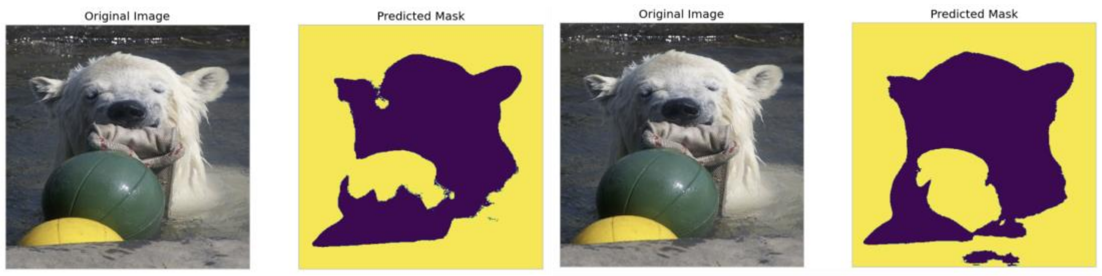


Figure 5 Testing result

(左) UNet++with dpn68b 和 (右) UNet++ with dpn68b / dpn92 / efficientnet-b2

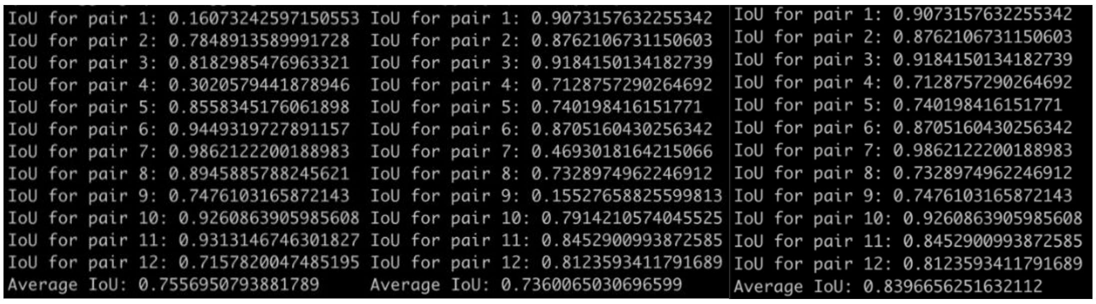
4 Testing data results and discussion

Testing data during the demo contains 12 images. The average IoU of our work in testing data:

- DL method: 75.57%
- Conventional method: 73.60%
- Combined (use DL in reflection scenes): 83.97%

In Figure 15, we can see some data cannot be efficiently segmented by the Conventional method that we propose in Chapter 2. Those are the reflection problems we discuss above. However, the same data can be processed by the DL method.

If we use two different methods, the shortcomings can be covered by each other. Replace those bad results in the conventional method with the results in the DL method. The average IoU can achieve 83.97%.



```
IoU for pair 1: 0.16073242597150553 IoU for pair 1: 0.9073157632255342  IoU for pair 1: 0.9073157632255342
IoU for pair 2: 0.7848913589991728  IoU for pair 2: 0.8762106731150603  IoU for pair 2: 0.8762106731150603
IoU for pair 3: 0.8182985476963321  IoU for pair 3: 0.9184150134182739  IoU for pair 3: 0.9184150134182739
IoU for pair 4: 0.3020579441878946  IoU for pair 4: 0.7128757290264692  IoU for pair 4: 0.7128757290264692
IoU for pair 5: 0.8558345176061898  IoU for pair 5: 0.740198416151771   IoU for pair 5: 0.740198416151771
IoU for pair 6: 0.9449319727891157  IoU for pair 6: 0.8705160430256342  IoU for pair 6: 0.8705160430256342
IoU for pair 7: 0.9862122200188983  IoU for pair 7: 0.4693018164215066  IoU for pair 7: 0.9862122200188983
IoU for pair 8: 0.8945885788245621  IoU for pair 8: 0.7328974962246912  IoU for pair 8: 0.7328974962246912
IoU for pair 9: 0.7476103165872143  IoU for pair 9: 0.15527658825599813 IoU for pair 9: 0.7476103165872143
IoU for pair 10: 0.9260863905985608 IoU for pair 10: 0.7914210574045525 IoU for pair 10: 0.9260863905985608
IoU for pair 11: 0.9313146746301827 IoU for pair 11: 0.8452900993872585 IoU for pair 11: 0.8452900993872585
IoU for pair 12: 0.7157820047485195 IoU for pair 12: 0.8123593411791689 IoU for pair 12: 0.8123593411791689
Average IoU: 0.7556950793881789     Average IoU: 0.7360065030696599     Average IoU: 0.8396656251632112
```

Figure 60 Testing IoU of testing data

(left) DL method, (middle) Conventional method, and (right) Combined method

5 Appendix

Besides the above method, we also try other methods that may not be better than we have discussed. But they may be efficient in some specific scenes.

5.1 Textural features segmentation

Extract textural features by a Gabor filter, then add to K-means features clustering. Efficient in some specific scenes that have obvious textures like sea waves.

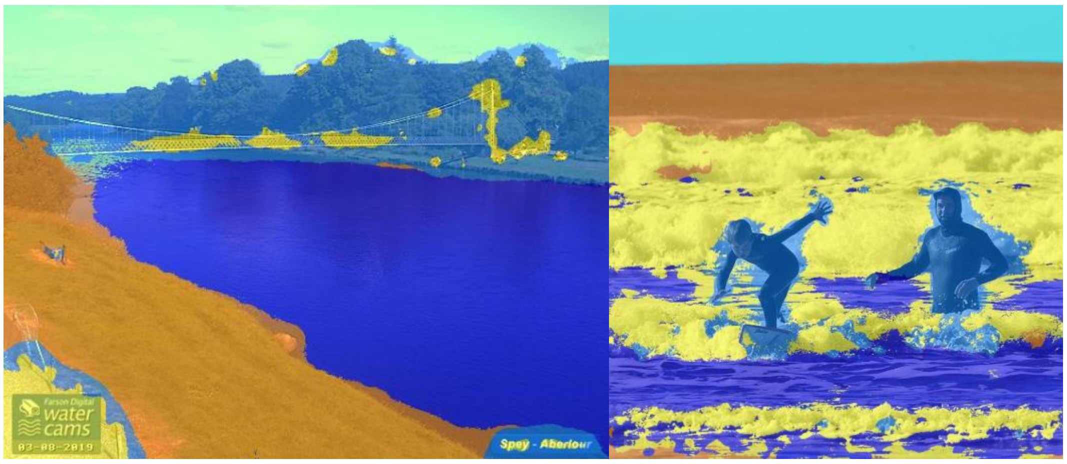


Figure 21 Textural feature segmentation result

6 Reference

[1] Transfer learning flow
https://medium.datadriveninvestor.com/what-you-must-know-about-transfer-learning-4a6e4cb9fbad

[2] Pytorch segmentation-model document
https://segmentation-modelspytorch.readthedocs.io/en/latest/#models

[3] Ensemble learning graph
https://towardsdatascience.com/ensembles-in-machine-learning-9128215629d1