SIMPLE PRESENTATION FOR COMPETITION UW-MADISON U-NET & U-NET++

UW-MADISON GI TRACT IMAGE SEGMENTATION

MINGZIRUI WU BOTAO JIANG

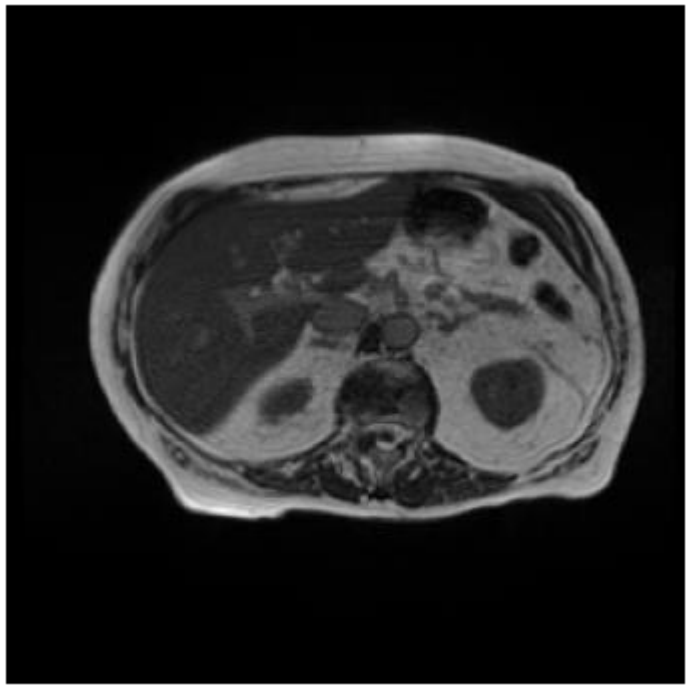
INVOLVED LIBRARY

- Albumentations
  - Albumentations is a computer vision tool that boosts the performance of deep convolutional neural networks.
- Segmentation Models
  - Python library with Neural Networks for Image Segmentation based on PyTorch.

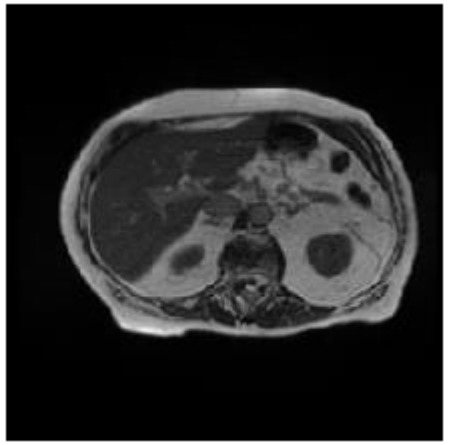
ALBUMENTATIONS

- The process of image augmentation
  1. Import albumentations and a library to read images from the disk
  2. Define an augmentation pipeline.
  3. Read images from the disk.
  4. Pass images to the augmentation pipeline and receive augmented images.

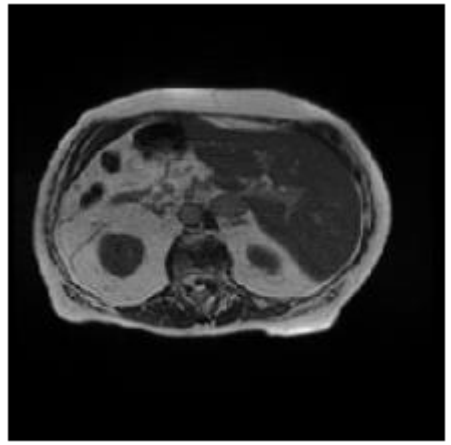
AUGMENTATION STRATEGY



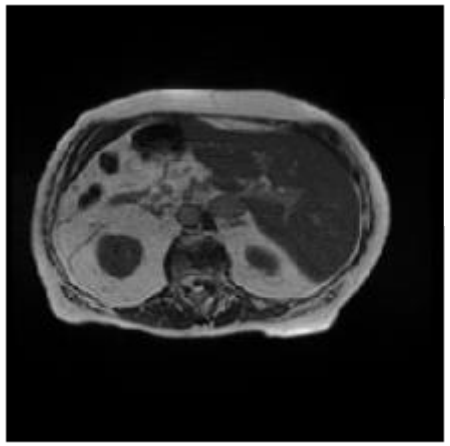
```
A.Resize([224, 224],
interpolation=cv2.INTER_NEAREST)
```



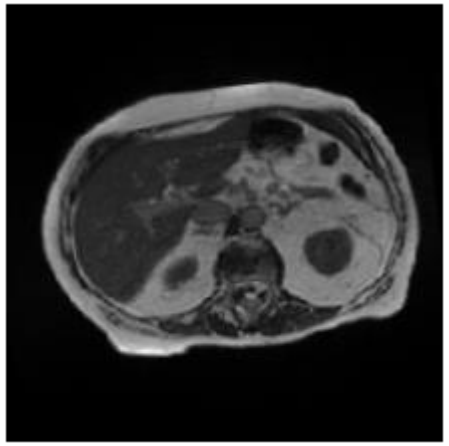
```
A.HorizontalFlip
(p=0.5)
```



INITIAL AUGMENTATION STRATEGY

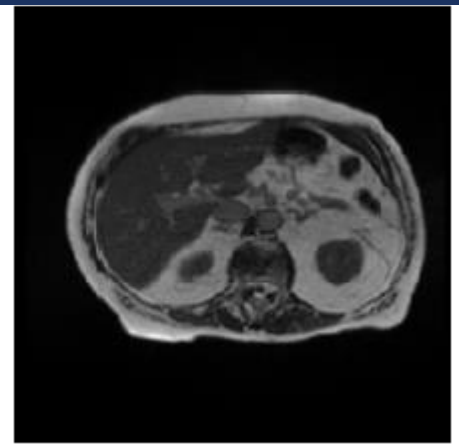


```
A.ShiftScaleRotate(shift_limit=0.0625, scale_limit=0.05, rotate_limit=10, p=0.5)
```

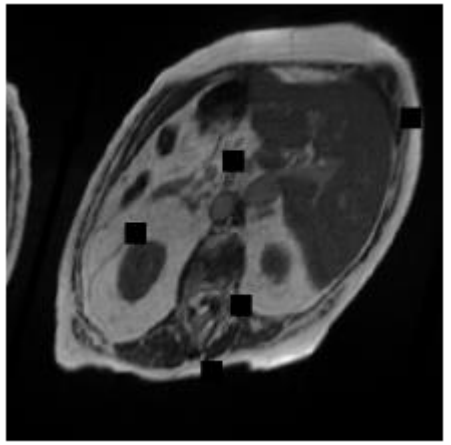


A.OneOf

```
A.GridDistortion(num_steps=5, distort_limit=0.05, p=1.0),
```

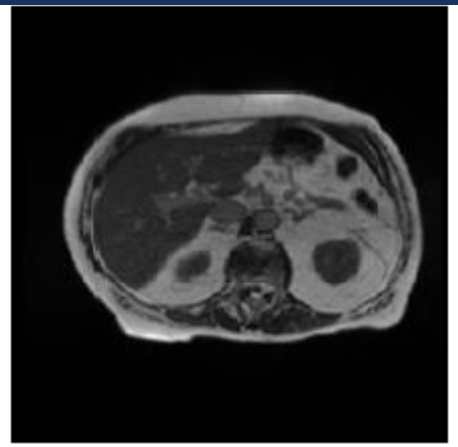


```
A.ElasticTransform(alpha=1, sigma=50, alpha_affine=50, p=1.0)
```

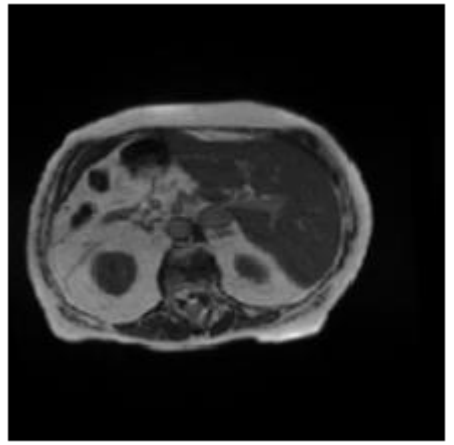


INITIAL AUGMENTATION STRATEGY

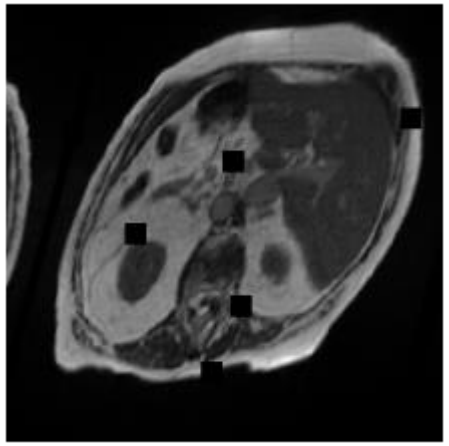
```
A.GridDistortion(num_steps=5,
distort_limit=0.05, p=1.0),
```

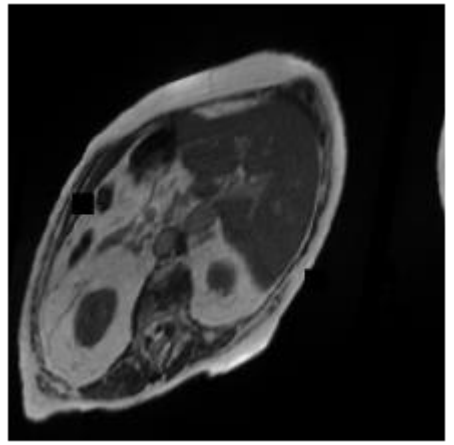


```
A.CoarseDropout(max_holes=8, max_height=CFG.img_size[0]//20, max_width=224//20,
                min_holes=5, fill_value=0, mask_fill_value=0, p=0.5),
```



```
A.ElasticTransform(alpha=1, sigma=50,
alpha_affine=50, p=1.0)
```





SEGMENTATION MODELS

- Unet
  - With imagenet for pre-trained weights: 0.838
- Unet++
  - With efficient-b1 for pre-trained weights: 0.840
- DeepLabV3
  - With efficient-b1/imagenet for pre-trained weights: Out of Memory! TOO much Parameters!
- R2UNet
  - With efficientb1/imagenet for pre-trained weights: Out of Memory! TOO much Parameters!

PS: Further Detail Behind.

LOSS FUNCTION

- JaccardLoss
  - Jaccard(x, y) = |intersection(x, y)| / |union(x, y)|.
- DiceLoss
  - DicsLoss = 1 – (2 |intersection(x, y)| / |X| + |Y|)
- BCELoss: (Used for binary classification, need sigmoid)
  - Binary Cross Entropy

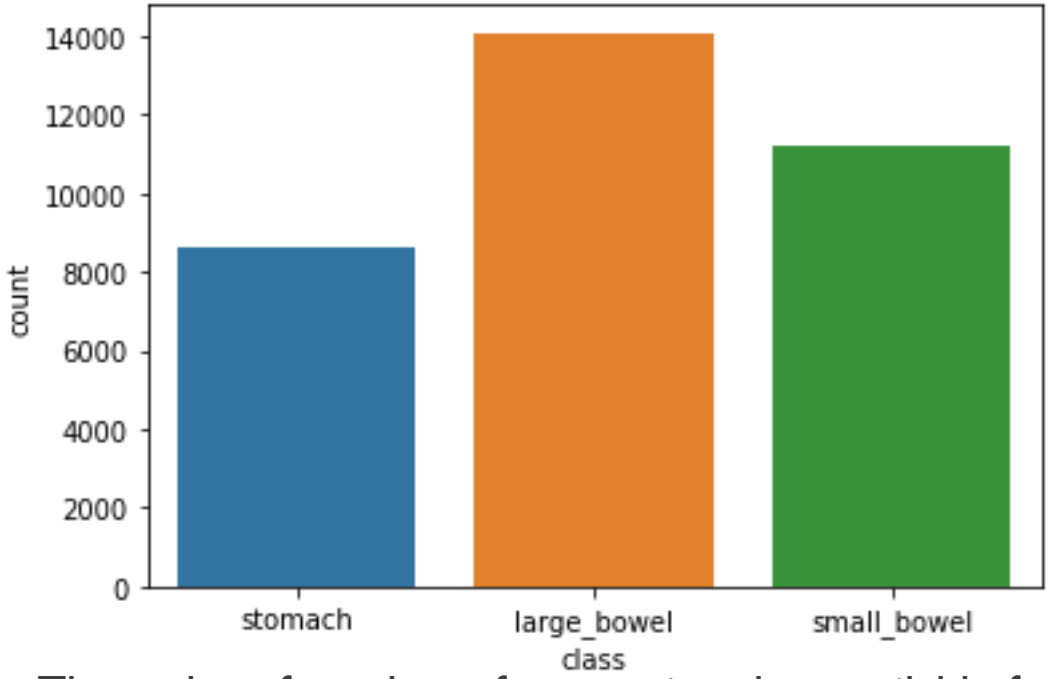
$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

- LovaszLoss
  - Better to use it in combination with cross entropy, or train the network with cross entropy first, and then use lovasz loss finetune.
- TverskyLoss
  - Designed to solve data imbalance in medical segmentation.

$$T(\alpha, \beta) = \frac{\sum_{i=1}^{N} p_{0i} g_{0i}}{\sum_{i=1}^{N} p_{0i} g_{0i} + \alpha \sum_{i=1}^{N} p_{0i} g_{1i} + \beta \sum_{i=1}^{N} p_{1i} g_{0i}}$$

LOSS FUNCTION

- Criterion
  - 0.5*BCELoss(y_pred, y_true) + 0.5*TverskyLoss(y_pred, y_true)
  - 0.6*BCELoss(y_pred, y_true) + 0.4*Dice(y_pred, y_true)
- Overlapping Mask
  - BCELoss
- Imbalance Data
  - TverskyLoss
  - Dice



- The order of number of annotation data available for the training can be observed from above.

PRACTICE

DIFFERENT MODEL AND DIFFERENT PRETRAINED DATASET

CURRENT RESULTS

Submission and Description	Status	Public Score
UWMGI: Unet [Infer] [PyTorch] Unet++ with Efficientnet_b1_success_2 (version 8/8) 21 hours ago by KaggleJbt Notebook UWMGI: Unet [Infer] [PyTorch] | Unet++ with Efficientnet_b1_success_2	Succeeded	0.840
UWMGI: Unet [Infer] [PyTorch] Unet_with_resnet34_success (version 5/8) a day ago by KaggleJbt Notebook UWMGI: Unet [Infer] [PyTorch] | Unet_with_resnet34_success	Succeeded	0.823
UWMGI: Unet [Infer] [PyTorch] Unet++ with Resnet34 new edition (version 3/8) 2 days ago by KaggleJbt Notebook UWMGI: Unet [Infer] [PyTorch] | Unet++ with Resnet34 new edition	Succeeded	0.545
UWMGI: Unet [Infer] [PyTorch] Unet++ with Resnet34 version 1 (version 1/8) 3 days ago by KaggleJbt Notebook UWMGI: Unet [Infer] [PyTorch] | Unet++ with Resnet34 version 1	Succeeded	0.431
UWMGI: Unet [Infer] [PyTorch] (version 1/1) 3 days ago by cenxun Notebook UWMGI: Unet [Infer] [PyTorch] | Version 1	Succeeded	0.838
UWMGI: Unet [Infer] [PyTorch] Unet++ Efficientnet-b1 Mixup Dice+CE (version 9/9) 7 hours ago by KaggleJbt Notebook UWMGI: Unet [Infer] [PyTorch] | Unet++ Efficientnet-b1 Mixup Dice+CE	Succeeded	0.833

Four valid results:

- Unet with resnet34: 0.823
- Unet with EfficientNet-b1: 0.838
- Unet++ with EfficientNet-b1: 0.840
- Unet++ with EfficientNet-b1 applying Mixup: 0.833

HYPER PARAMETERS

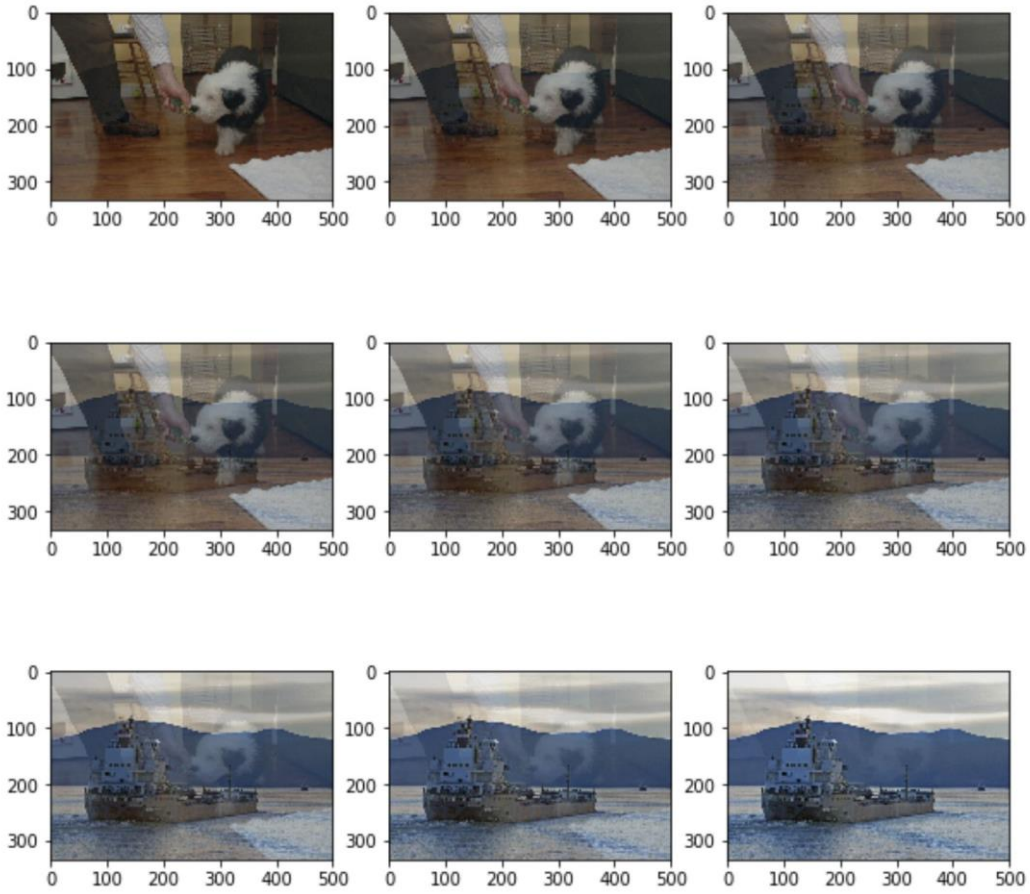
- Scheduler: CosineAnnealingLR
- Optimizer: Adam (lr = 2e-3, weight_decay = 1e-6)
- train_bs: 64
- valid_bs: train_bs*2
- img_size: [224, 224]
- Epochs: 15

DATA AUGMENTATION: MIXUP

- [1710.09412] mixup: Beyond Empirical Risk Minimization (arxiv.org)

- Mixup trains a neural network on convex combinations of pairs of examples and their labels.

DATA AUGMENTATION: MIXUP



1. For a batch of images to be tested, we fuse one of them with another randomly selected image in the same batch with the fusion ratio Lam to obtain mixed tensor inputs.

2. Lam is a random real number between [0,1], which conforms to the beta distribution. When adding, pixel values corresponding to the two images are directly added.

3. Transfer the mixed inputs to the model to obtain the outputs. Secondly, we calculate the loss function for the labels of the two pictures respectively, and then calculate the weighted sum of the loss function according to the proportion Lam.

DATA AUGMENTATION: MIXUP

```
    for step, (images, masks) in pbar:
        images = images.to(device, dtype=torch.float)
        masks  = masks.to(device, dtype=torch.long)

        batch_size = images.size(0)

        alpha = CFG.alpha
        lam = np.random.beta(alpha, alpha)
        index = torch.randperm(images.size(0)).cuda()
        inputs = lam*images + (1-lam)*images[index,:]
        targets_a, targets_b = masks, masks[index]

        with amp.autocast(enabled=True):
            outputs = model(inputs)
            loss = lam * criterion(outputs, targets_a) + (1 - lam) * criterion(outputs, targets_b)
#             y_pred = model(images)
#             loss   = criterion(y_pred, masks)
            loss   = loss / CFG.n_accumulate

        scaler.scale(loss).backward()
```

FUTURE PLAN

- Try Different combinations of loss functions ASAP.
- Find more effective image transformation strategies.
- Tune the hyperparameters on the basis of Unet++.

Thank you!