AI 3603 Artificial Intelligence: Principles and Techniques

By: 521030910064, 521030910065, 521030910066, 521030910067

HW#: 3

January 4, 2024

I. INTRODUCTION

Segment Anything Model (SAM) is derived from the Segment Anything (SA) project of the Meta in 2023. Different from the general image segmentation algorithm, SAM aims to design a basic model of image segmentation that can segment even the class of objects that have not been seen in the training phase, that is, zero-shot. In this project, we will use SAM and a specially designed model MedSAM (pre-trained for medical uses) to segment the tumor area from the MRI images of brains.

SAM allows different kinds of prompts, for which we divided it into the two tasks. In task1, we uses the "box prompt" to do the segmentation and realizes the evaluation module, instead task2 focus on "point prompt" generating by masks.

II. TASK 1

Firstly, we use the file `pre_grey_rgb2D.py` to transform image directory to npz file, and normalize the image size from 515*512 to 256*256.

Then use `finetune_and_inference_tutorial_2D_dataset.ipynb` to do the training and testing, the file provides a full steps flow to load data, train, test, and visualize results.

A. Choose boxes

As the image shown below, the box is presented by a blue rectangle noting where should be focused and separated. We have masks with each image, so it's available to attain the minimum rectangle which just contains the mask, just as the following shown.

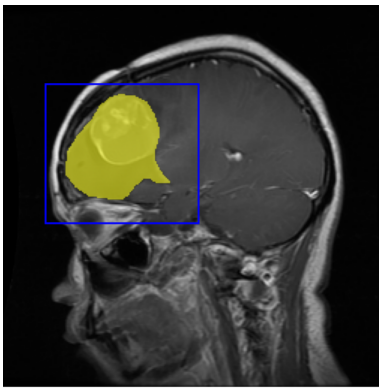

FIG. 1: The box marks the target.

```
def __getitem__(self, index):
        img_embed = self.img_embeddings[index]
        gt2D = self.ori_gts[index]
        y_indices, x_indices = np.where(gt2D > 0)
        x_min, x_max = np.min(x_indices), np.max(x_indices)
        y_min, y_max = np.min(y_indices), np.max(y_indices)
        H, W = gt2D.shape

        # ADD SOME RANDOM BIAS
        x_min = max(0, x_min - np.random.randint(0, 20))
```

```
        x_max = min(W, x_max + np.random.randint(0, 20))
        y_min = max(0, y_min - np.random.randint(0, 20))
        y_max = min(H, y_max + np.random.randint(0, 20))
        bboxes = np.array([x_min, y_min, x_max, y_max])
        # CONVERT IMG EMBEDDING, MASK, BOUNDING BOX TO TORCH TENSOR
        return torch.tensor(img_embed).float(), torch.tensor(gt2D[None, :,:]).long(), torch.tensor(bboxes
            ).float()
```

B. Training

The general structure of training code is given. The key is to find a proper function to optimize (with the learning rate and weight loss). After a few attempt, we choose `lr = 1e-4` and `wd = 1e-3` with Adam optimizer. The specific parameters we used is below:

```
optimizer = torch.optim.Adam(sam_model.mask_decoder.parameters(), lr=1e-4, weight_decay=0.001)
```

After 1,000-epoch training, the Dice&Cross Entropy loss gets lower than 0.05.

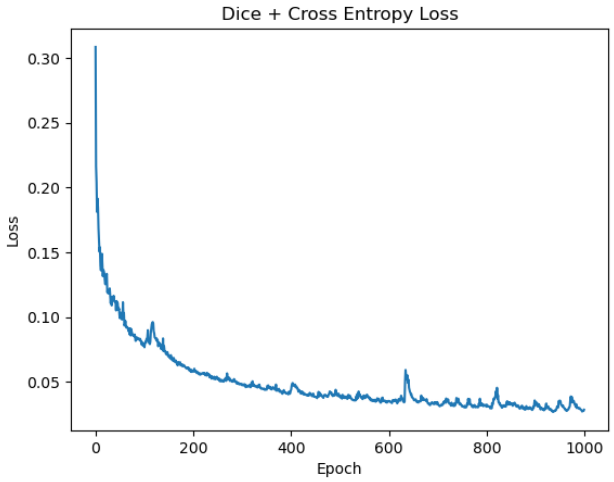

FIG. 2: The Dice&Cross Entropy loss during 1,000 epochs.

C. Testing

The task is required to use IOU to evaluate the quality of SAM. To calculate IOU, we introduce another value DSC (dice similarity coefficient). The relation between the two values can be expressed by

$$IOU = \frac{DSC}{2 - DSC}.$$

```
def dsc2iou(i):
    return i / (2 - i)
ori_sam_dsc = compute_dice_coefficient(gt_data>0, ori_sam_seg>0)
```

Then run the `ori_sam` and medSAM models on the test dataset using the same box before, the result is shown at FIG 3. The median IOU is 0.8667.

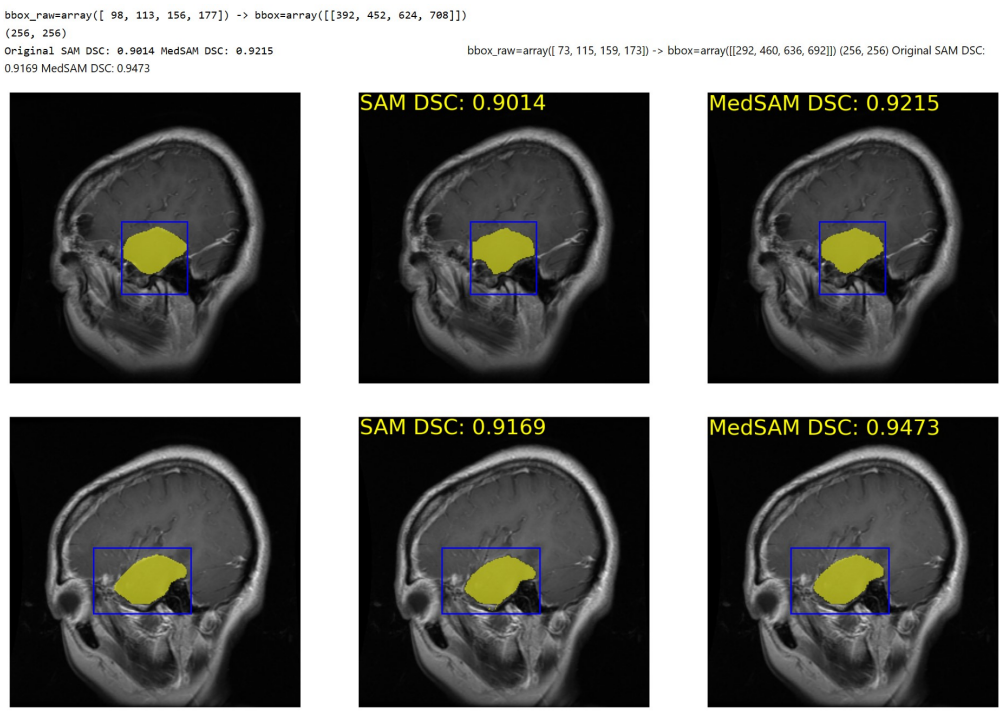

FIG. 3: A piece of the testing result and the IOU score.

III. TASK 2

Task 2 requires using a prompt point to replace the box. We modify the prompt firstly. As the code shown, `y_indices, x_indices` contains all points in the mask, and it's quite hard to take all these points into account. Then to pick up one point that can represent all of the others, we take the points in 2-dimension as a graph and choose the center point of mass. It can be see better than take other points from the result.

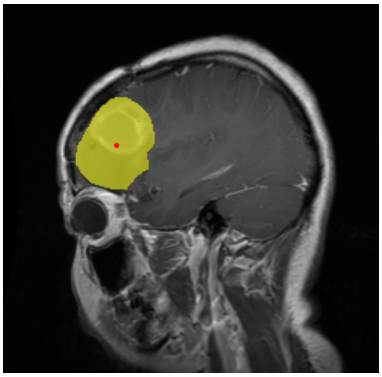

FIG. 4: The red point represents the whole highlighted area.

```
def __getitem__(self, index):
        img_embed = self.img_embeddings[index]
        gt2D = self.ori_gts[index]
        y_indices, x_indices = np.where(gt2D > 0)
        # Calculate the index of center of mass point
        x_center = np.mean(x_indices)
        y_center = np.mean(y_indices)
        points = np.array([x_center, y_center])
        # Returns the image embeddings, truth-labels and the center of mass point
        return torch.tensor(img_embed).float(),
               torch.tensor(gt2D[None, :,:]).long(),
               torch.tensor(points).float()
```

Usually different with boxes, the point prompt need a set of points of a label, which describes the types of points. Here we only set one point for each image, so all the label should be set equal. We pass a `Tuple([Batchsize, 1, 2], [Batchsize, 1])` at last to prompt encoder.

```
with torch.no_grad():
    point_np = point.numpy()
    sam_trans = ResizeLongestSide(sam_model.image_encoder.img_size)
    point_np = sam_trans.apply_coords(point_np, (gt2D.shape[-2], gt2D.shape[-1]))

    labels_np = np.zeros((point_np.shape[0], 1), dtype=int)

    points_tensor = torch.as_tensor(point_np, dtype=torch.float32, device=device)
    labels_tensor = torch.as_tensor(labels_np, dtype=torch.int64, device=device)

    if len(points_tensor.shape) == 2:
        points_tensor = points_tensor[:, None, :] # (B, 1, 4)

    points_tuple = (points_tensor, labels_tensor)
    print(points_tensor.shape)
    print(labels_tensor.shape)

    sparse_embeddings, dense_embeddings = sam_model.prompt_encoder(
        points=points_tuple,
        boxes=None,
        masks=None,
    )
```

The loss fluctuates around 0.8 (much more than prompt of box). This result indicates that providing only one point maybe not enough in this complex situation. Using the same changes as box in the testing, we change the parameter converting to model and the result is as followed. The median IOU is 0.47 and the result presents a relatively extreme situation like IOU be larger than 0.90 or lower than 0.10. It claims that single-point prompt usually works well for a block mask, however if the mask is complex, single-point prompt MedSAM cannot find the full mask.

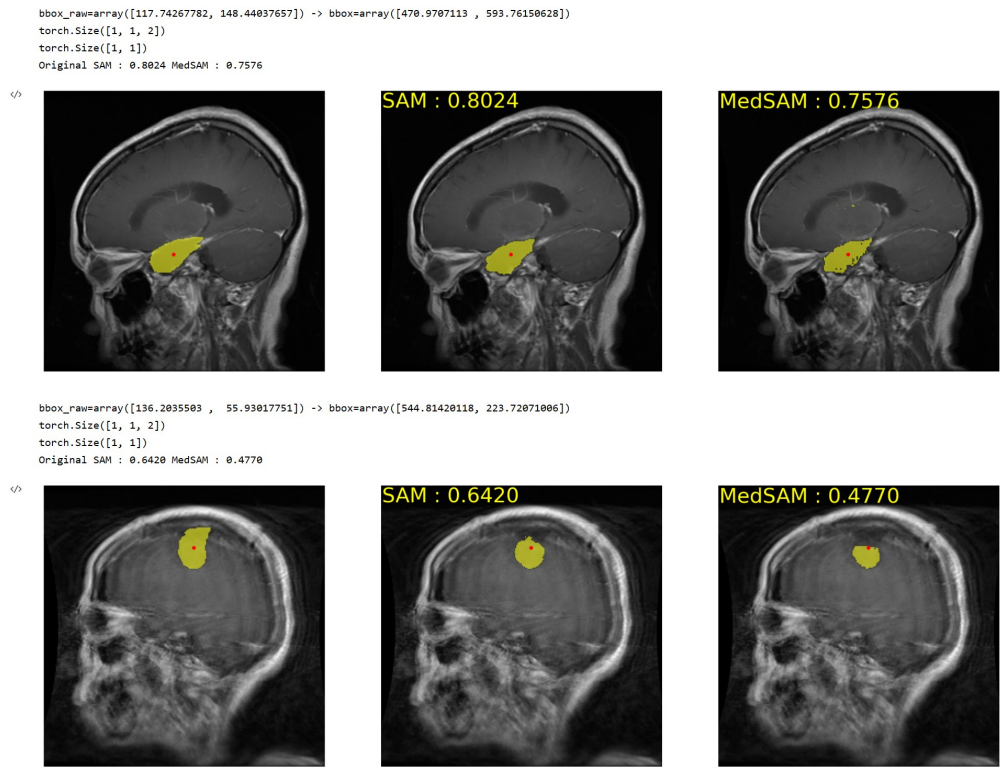

FIG. 5: The red point represents the whole highlighted area.

IV. COMPARISON AND CONCLUSION

The two prompts not only indicate the different methods to do the segmentation tasks, but in reality, they're often employed in SAM by combination. The different advantages of the two prompts are below:

- Points are useful for capturing local dependencies and generating coherent sequences. Boxes represents a global context or summary of the entire input sequence, allowing the model to capture long-range dependencies and contextual information.
- Points are suitable for capturing local dependencies and sequential patterns, while the box helps capture global context and long-range dependencies.
- The combination of points and box allows the SAM model to balance efficiency and coherence in sequence generation tasks.

In this homework, only use one prompt leads to the lack of efficiency, and the point of prompt is more worse for our over-simplification. In the future the point can be choose by getting more points to represent the MRI area.