AI 3603 Artificial Intelligence: Principles and Techniques

By: Shen Junzhe (521030910108), Li Jiaxin (521030910096), Huang Benhao (521030910073)

HW#: 3

January 4, 2024

Contents

I. INTRODUCTION

In this homework, we finetune the Segment Anything model(SAM)[1] to segment the tumor area from the MRI images of brains. We use two kinds of prompts: boxes and points.

II. SAM

SAM has three components, illustrated in FIG. 1: an image encoder, a flexible prompt encoder, and a fast mask decoder.

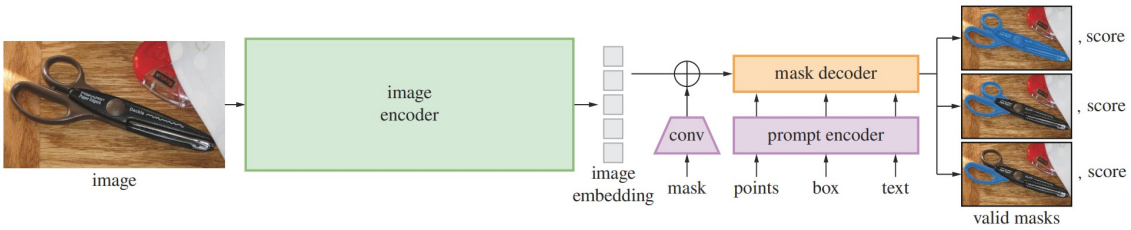

FIG. 1: Segment Anything(SAM) overview. A heavyweight image encoder outputs an image embedding that can then be efficiently queried by a variety of input prompts to produce object masks at amortized real-time speed. For ambiguous prompts corresponding to more than one object, SAM can output multiple valid masks and associated confidence scores.

A. Image Encoder

In general, the image encoder can be any network that outputs a $C \times H \times W$ image embedding. Motivated by scalability and powerful pretraining methods, an MAE pre-trained Vision Transformer (ViT)[2] with minimal adaptations is used to process high resolution inputs. The image encoder runs once per image and can be applied prior to prompting the model.

Following standard practices, an input resolution of 1024×1024 obtained by rescaling the image and padding the shorter side is adopted. The image embedding is therefore 64×64. To reduce the channel dimension, a 1×1 convolution is used to get 256 channels, followed by a 3×3 convolution also with 256 channels. Each convolution is followed by a layer normalization.

B. Prompt Encoder

Two sets of prompts: *sparse* (points, boxes, text) and *dense* (masks) are supported.

1. *Sparse prompt*

Sparse prompts are mapped to 256-dimensional vectorial embeddings as follows. A point is represented as the sum of a positional encoding of the point's location and one of two learned embeddings that indicate if the point is either in the foreground or background.

A box is represented by an embedding pair: (1) the positional encoding of its top-left corner summed with a learned embedding representing "top-left corner" and (2) the same structure but using a learned embedding indicating "bottom-right corner".

Finally, to represent free-form text, the text encoder from CLIP[3] is used (any text encoder is possible in general).

2. *Dense prompt*

Dense prompts (i.e., masks) have a spatial correspondence with the image. The input mask is at a $4\times$ lower resolution than the input image, then downscaled an additional $4\times$ using two 2×2, stride 2 convolutions with output channels 4 and 16, respectively. A final 1×1 convolution maps the channel dimension to 256. Each layer is separated by GELU activations and layer normalization. The mask and image embedding are then added element-wise. If there is no mask prompt, a learned embedding representing "no mask" is added to each image embedding location.

C. Lightweight Mask Decoder

This module efficiently maps the image embedding and a set of prompt embeddings to an output mask. The architecture is shown in FIG. 2.

Each decoder layer performs 4 steps: (1) self-attention on the tokens, (2) cross-attention from tokens (as queries) to the image embedding, (3) a point-wise MLP updates each token, and (4) cross-attention from the image embedding (as queries) to tokens. This last step updates the image embedding with prompt information. During cross-attention, the image embedding is treated as a set of 64^2 256-dimensional vectors. Each self/cross-attention and MLP has a residual connection, layer normalization, and a dropout of 0.1 at training. The next decoder layer takes the updated tokens and the updated image embedding from the previous layer.

III. TASK1: PROMPTING THE MODEL IN BOXES

Boxes are represented in 4-tuples in the form of `[x_min, y_min, x_max, y_max]`. Choosing a box to prompt the model is very important to the performance of SAM models. In practice, humans don't have a precise control when choosing a box prompt. Thus it is necessary to introduce randomness into the choice of box.

We assume we always have access to the ground truth mask when choosing the prompts, otherwise it would be nearly impossible to choose a proper prompt for the computer without any prior knowledge.

We set the baseline boxes to be those whose lines are tangent to the mask. The boxes can be expanded or shrinked by a random margin. An illustration is provided in FIG. 3.

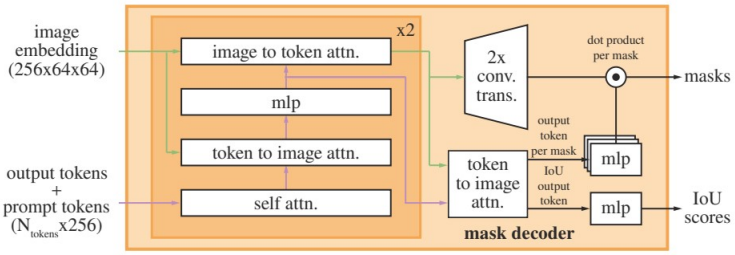

FIG. 2: Details of the lightweight mask decoder: A two-layer decoder updates both the image embeddings and prompt tokens via cross-attention. The image embedding is then upscaled, from which the updated output tokens are used to dynamically predict masks. (Not illustrated for figure clarity. At every attention layer, positional encodings are added to the image embedding, and the entire original prompt token(including position encoding), is re-added to the token queries and keys.)

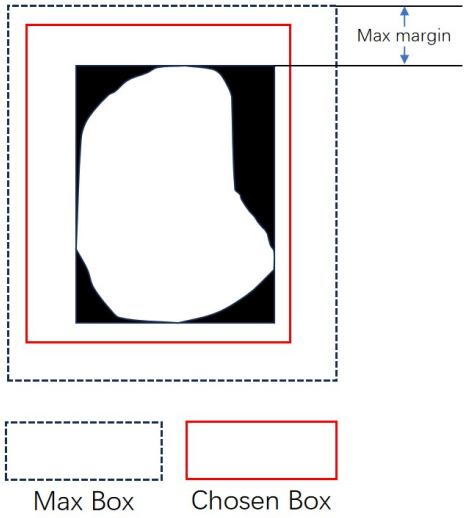

FIG. 3

To evaluate how the choice of boxes affects segmentation quality, we adjusted the max margin value in a range from -20 to 40, using increments of 10. The experiment results on both test set and train set are presented in FIG. 4 and FIG. 5.

Firstly, it is observed that baseline boxes (characterized by lines tangential to the mask) results in the optimal Intersection over Union (IOU) scores. Furthermore, there is a trend where the IOU scores increase in correspondence with the enlargement of box size when the max margin size is less than 0. Conversely, the IOU scores demonstrate a decrease in relation to the enlargement of box size when the max margin size exceeds 0. Also, the finetuned model performing worse on the test set when the max margin size exceeds 10 suggests a decline in its generalization ability as the max margin size becomes too big.

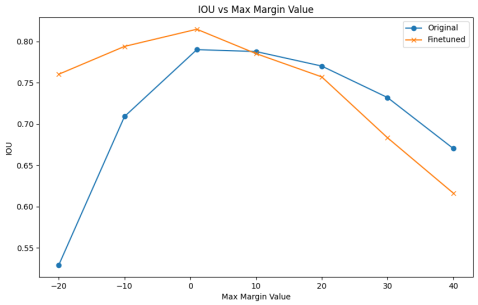

FIG. 4: Test set

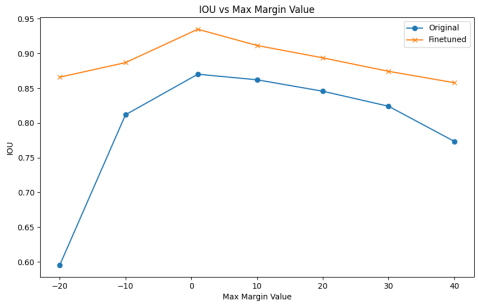

FIG. 5: Train set

Thus, the selection of a box of appropriate size is crucial for achieving desirable IOU scores.

IV. TASK2: PROMPTING THE MODEL IN POINTS

Points are simply represented in the form of `(x, y)`. There are two kinds of point prompts, one is foreground point and the other is background point. A background point is used to filter out the background region from the foreground.

We also assume that we have access to the ground truth mask when choosing the point prompts.

A. Random Choice

1. *Random in the box*

One point is chosen randomly inside the baseline box of a mask, labeled as foreground point. We refer to this approach as *random_box*.

2. *Random in the mask*

One point is chosen randomly inside the mask, labeled as foreground point. We refer to this approach as *random_mask*.

3. *Random in the mask with Gaussian*

Inspired by the fact that human is tend to chose the center of target but it is impossible to chosen the center totally accurately , we simulate the process of choice point prompt using random choice with the Gaussian weight:

$$m_k = G_\sigma(\|(x_k, y_k) - (\bar{x}, \bar{y})\|_2^2)$$

One point is chosen randomly inside the mask with the Gaussian weight and labeled as foreground point. We refer to this approach as *random_gaussian*.

B. Center Choice

1. Center of Box

One point is chosen at the center of the baseline box, labeled as foreground point. `[(x_min+x_max)/2, (y_min+y_max)/2]`. We refer to this approach as *center_box*.

2. Center of Mass I

One point is chosen at the center of mass of the mask, labeled as foreground point. The center of mass in discrete 2D space is defined as

$$(\frac{\sum_k m_k x_k}{\sum_k m_k}, \frac{\sum_k m_k y_k}{\sum_k m_k})$$

where m_k indicates the mass of the kth point. In this case, m_k is set to the mask value of the kth point. We refer to this approach as *center_mask*.

3. Center of Mass II

Inspired from bilateral filtering, a well known technique in digital image processing, instead of setting m_k to the mask value, we set the mass for each pixel to

$$m_k = G_\sigma(\|(x_k, y_k) - (\bar{x}, \bar{y})\|_2^2)$$

where G_σ stands for the PDF of Gaussian distribution with variance σ, $(\bar{x}, \bar{y})$ is the coordinate of the center of the baseline box. . We refer to this approach as *center_gaussian*.

C. Background Prompt

One point is chosen at the corner of the box (to make sure it does not lie in the tumor region), labeled as background point.

D. Experiment Result

1. Result Display

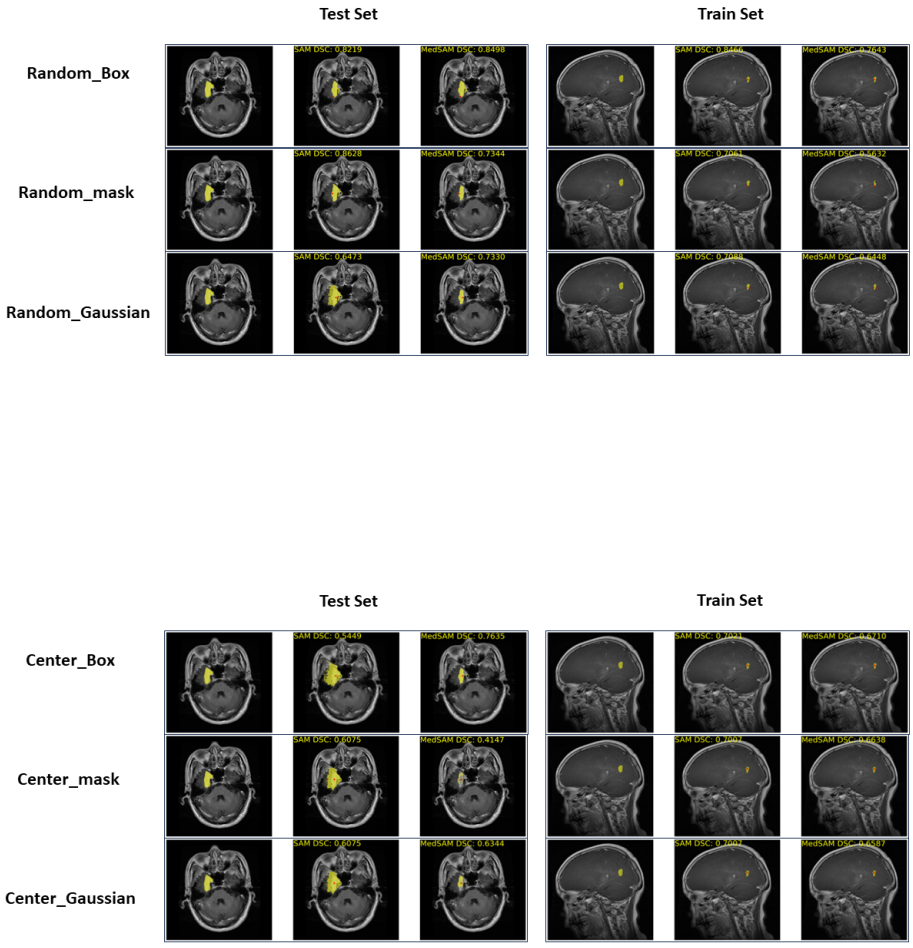

FIG. 6: The Result of different foreground point prompt.

2. Result Analysis Comparison between box prompt and box prompt

Through comparing the impact of the box prompt and the point prompt on the segmentation performance of the SAM model, we observed that the box prompt had a more significant effect on improving the segmentation performance. This result may be attributed to the characteristics of the two prompts. When using the mask boundary as the box, the model tends to directly extract the segmented regions within the box that exhibit large areas of similar colors. On the other hand,

		Test Set		Train Set	
		origin	finetuned	origin	finetuned
random	box	0.4612	0.4773	0.6183	0.6560
	mask	0.5308	0.4841	0.7284	0.7614
	gaussian	0.5703	0.4795	0.7259	0.7623
center	box	0.5827	0.6309	0.7629	0.7766
	mask	0.5905	0.6302	0.7634	0.7750
	gaussian	0.5911	0.6313	0.7637	0.7751
background		0.0815	0.0753	0.1856	0.2386

TABLE I: The IoU Score using different point prompt [a]

[a]We find the result in this experiment varies from different device. We think it is possible that the device set will make a difference to the result though we have set the same random seed. Finally, we select the result from the group memeber Shen Junzhe.

the point prompt guides the model to segment regions with similar colors, but lacks spatial guidance in comparison.

3. Result Analysis : Different Point Prompt

Firstly, when comparing the results of different methods of random choice, it is observed that the performance is better when foreground points are selected randomly within the box rather than within the mask. This outcome is somewhat counterintuitive, as the performance in the training set is inferior. Given the limited scale of the dataset, this discrepancy may be attributed to potential overfitting, evidenced by the decreased score following fine-tuning. Additionally, beyond the performance in the test set post-fine-tuning, it was found that randomly selecting points on the mask and integrating Gaussian weights for random point selection yielded performance enhancements. These methods surpassed the results achieved through the random selection of points on the box.

Secondly, when comparing various random point selection methods, it is evident that several center-point selection approaches consistently exhibit significant performance improvements on both the test and validation sets. This holds true irrespective of their application before or after fine-tuning. Concurrently, it is important to highlight that center-point selection effectively counters the "overfitting" phenomenon frequently associated with random point selection methods.

Lastly, through the comparison of various random point selection methods, it has been observed that the assignment of Gaussian weights to the mask, followed by the selection of weighted center points, enables the algorithm to achieve an equilibrium between the center of the box and the centroid of the mask. This strategy notably improves the model's generalization capability to a certain degree. Consequently, this approach yields the optimal performance on the test set.

By the way, we make the exploration experiment about the background point prompt . As the table illustrated, its performance is worse than the foreground point prompt. We think it works only when combined with foreground point because the background point prompt is used to delete the uninterested region.

V. FUTURE RESEARCH METHOD

Due to time constraints and the limitations of the given code framework, the point setting during the training process only considers the mask without taking the original image into account. Here, we propose a method that may significantly improve performance and requires further experimental exploration.

A. Observations

In some data samples, we have noticed certain color variations in the CT images within the mask region. We speculate that this could be attributed to labeling errors, where the mask region also includes normal areas. Consequently, relying solely on the mask to infer the points for training prompts may introduce errors.

B. Method Description

If we can consider both the mask and the original image, we can assign higher weights to large areas within the mask that have similar colors, while assigning lower weights to regions with significant color differences. This approach allows us to leverage both random point selection and center point selection methods to solve the problem effectively.

The formulation of weight is as follow:

$$m_k = G_{\sigma_s}(\|(x_k, y_k) - (\bar{x}, \bar{y})\|_2^2) G_{\sigma_r}(\|I_{(x_k, y_k)} - M(I)\|)$$

$M(I)$: the most frequently occurring color
$G_{\sigma(r)}$: control the scale of color which make senses
$G_{\sigma(s)}$: trade off between the center of the box and the centriod of the mask

VI. CONCLUSION

In this experiments:

- We familiarized ourselves with the state-of-the-art model SAM in the field of image segmentation. We explored its fundamental architecture and usage. Subsequently, we fine-tuned the SAM model using the provided dataset and evaluated its performance on the task of cancer region segmentation.
- We also investigated the impact of different box sizes on the segmentation performance of the SAM model.
- By comparing the differences in segmentation performance between different prompts, we further analyzed the distinct characteristics of prompts in guiding region segmentation.
- Lastly, we also explored different prompt configuration methods. We proposed a centroid approach with Gaussian weights, which achieved a balance between the center of the box and the centroid of the mask. This approach improved the model's generalization to some extent.

A. Appendix

B. Appendix 1: The implement of get box

```
def get_bbox_from_mask(mask):
    '''Returns a bounding box from a mask'''
    y_indices, x_indices = np.where(mask > 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)
    # add perturbation to bounding box coordinates
    H, W = mask.shape
    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))

    return np.array([x_min, y_min, x_max, y_max])
```

C. Appendix 2: The implement of get point

```
from typing import Literal
def get_points(box, mode:Literal['middle', 'random', 'margin', 'centroid', 'gaussian','random_mask',"
    random_gaussian"]='middle', mask=None):
    if mode == 'middle':
        return [
            (box[0].item() + box[2].item()) / 2,
            (box[1].item() + box[3].item()) / 2
        ]
    elif mode == 'random':
        return [np.random.randint(box[0].item(), box[2].item()),
                np.random.randint(box[1].item(), box[3].item())]

    elif mode == "random_mask":
        assert mask is not None
        if isinstance(mask, torch.Tensor):
            mask = torch.squeeze(mask)
            ones = mask.nonzero()


            if ones.size(0) == 0:
                return None

            # Randomly select one index from the ones tensor
            random_index = torch.randint(0, ones.size(0), (1,)).item()
            random_point = ones[random_index]

            # Return the randomly sampled point as a list of integers
            return [int(random_point[0].item()), int(random_point[1].item())]
        elif isinstance(mask, np.ndarray):
            mask = np.squeeze(mask)
            y , x = mask.nonzero()


            random_index = np.random.randint(0, y.size, (1,)).item()
            random_point = (x[random_index] , y[random_index])
            return [int(random_point[0]) , int(random_point[1])]

    elif mode == "random_gaussian":
        assert mask is not None

        if isinstance(mask, torch.Tensor):
            gaussian_func = lambda d_2 : (-0.5 * d_2 / 1e4).exp() # Here we ignore the constant because
                it does not make it difference
            mask = torch.squeeze(mask).to(device)
            ones = mask.nonzero().to(device)
            # print("non_zeros:", ones)

            if ones.size(0) == 0:
                return None

            # Calculate distances from the center for all non-zero points
            center = torch.tensor([ (box[0].item() + box[2].item()) / 2 ,  (box[1].item() + box[3].item()
                ) / 2]).to(device)
            distances = torch.norm(ones - center, dim=1).to(device)

            # Calculate weights based on distances
            weights = gaussian_func(distances).to(device) # Example: using a simple Gaussian distribution
```

```
            # Normalize weights to probabilities
            probabilities = weights / weights.sum()

            # Randomly sample one index based on the probabilities
            selected_index = torch.multinomial(probabilities, 1).item()
            selected_point = ones[selected_index]
            # print("select:",selected_point)
            # print("center:",center)

            return [selected_point[0].item(), selected_point[1].item()]
        elif isinstance(mask, np.ndarray):
            mask_tensor = torch.from_numpy(mask).T.to(device)
            return get_points(box, mode, mask_tensor)



    elif mode == 'margin':
        rand_choice = np.random.randint(0, 4)
        if rand_choice == 0:
            return [box[0], box[1]]
        elif rand_choice == 1:
            return [box[0], box[3]]
        elif rand_choice == 2:
            return [box[2], box[1]]
        elif rand_choice == 3:
            return [box[2], box[3]]
        else:
            assert False
    elif mode == 'centroid':
        assert mask is not None
        if isinstance(mask, torch.Tensor):
            mask = torch.squeeze(mask)
            ones = mask.nonzero()
            center = torch.mean(ones, dim=0, dtype=torch.float32)
            return [int(center[0].item()), int(center[1].item())]
        elif isinstance(mask, np.ndarray):
            mask = np.squeeze(mask)
            y, x = mask.nonzero()
            x_mean = np.mean(x)
            y_mean = np.mean(y)
            return [int(x_mean.item()), int(y_mean.item())]
        else:
            raise ValueError(f'mask_has_type_{type(mask)},_which_is_not_supported')
    elif mode == 'gaussian':
        assert mask is not None
        if isinstance(mask, torch.Tensor):
            mask = torch.squeeze(mask).to(device)
            center_x, center_y = (box[0].item() + box[2].item()) / 2,\
                                 (box[1].item() + box[3].item()) / 2
            gaussian_func = lambda d_2 : (-0.5 * d_2 / 1e4).exp() # Here we ignore the constant because
                it does not make it difference
            len_x, len_y = mask.size()
            x_grid, y_grid = torch.meshgrid(torch.arange(len_x), torch.arange(len_y), indexing='ij')
            x_grid = x_grid.to(device)
            y_grid = y_grid.to(device)
            distance_square = ((x_grid - center_x)**2 + (y_grid - center_y)**2).to(device)
            gaussian = gaussian_func(distance_square).to(device)
            gaussian = gaussian * mask

            weighted_sum_x = (gaussian * x_grid).sum()
            weighted_sum_y = (gaussian * y_grid).sum()
            total_weight = gaussian.sum()

            center_of_mass_x = weighted_sum_x / total_weight
            center_of_mass_y = weighted_sum_y / total_weight
            return [int(center_of_mass_x.item()), int(center_of_mass_y.item())]
        elif isinstance(mask, np.ndarray):
            mask_tensor = torch.from_numpy(mask).T.to(device)
            return get_points(box, mode, mask_tensor)
    else:
        raise NotImplementedError(f'mode:_{mode}_not_implemented')
num_points = 1
```


[1] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, et al., arXiv preprint arXiv:2304.02643 (2023).

[2] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al., arXiv preprint arXiv:2010.11929 (2020).

[3] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al., *Learning transferable visual models from natural language supervision* (2021), 2103.00020.