6004 assignment-1

Tooth segmentation is a crucial task in oral medical image analysis, providing key technical support for applications such as orthodontic planning, dental disease diagnosis, and virtual implantation. In this assignment, students are required to design and implement a deep learning-based tooth segmentation algorithm to automatically segment target teeth from the provided 2D images, thereby developing their capabilities in medical image processing and segmentation algorithm design.

Task

For the given dataset, you need to train a semantic segmentation network for teeth. Each image in the dataset contains segmentation labels for a specific tooth ID, as well as pixel-level labels for the upper and lower gums (included in the mask image). Design a segmentation algorithm: basing on the existing data, take an RGB image of teeth and the ID of a specific tooth as input, and then segment the pixel region of that tooth.From the available tooth types in the dataset, at least **four different teeth** should be selected for training this segmentation task.

Final task requirement:

Given an **input image** and a **target tooth ID**, the model should output the pixel region corresponding to that tooth ID in the input image. And the **testing pseudocode:**

testing pseudocode

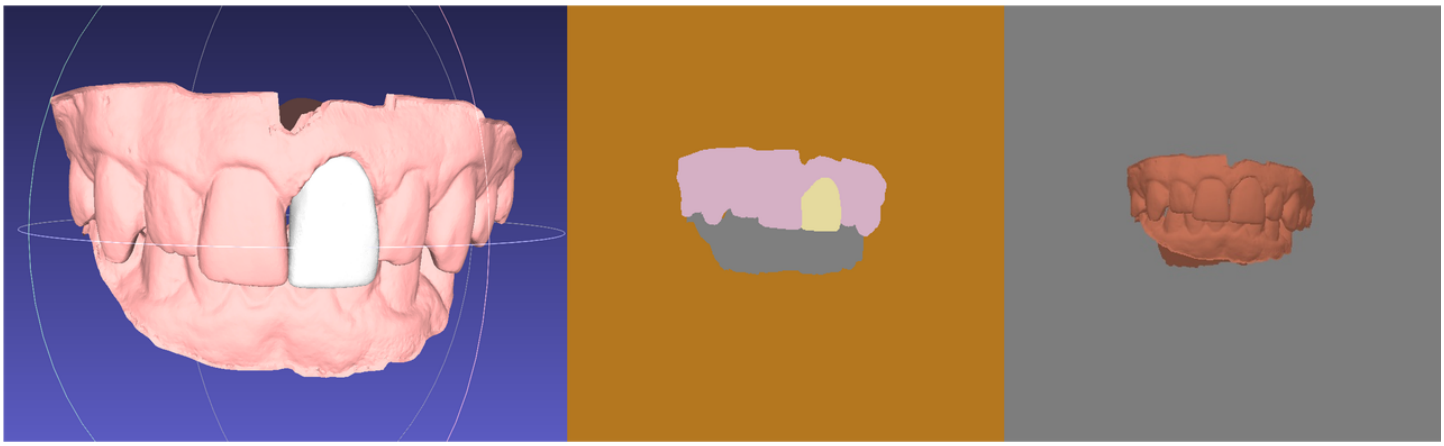
```
def Inference(input_image,tooth_id,model):
    mask_image = model(input_image,tooth_id)
    return mask_image
```

- Design an algorithmic pipeline to train and test a tooth segmentation network.
- **Write a complete report including:**
  - **Algorithm Design**
  - **Implementation Process** (e.g., network training setup, data preprocessing, etc.)
  - **Experimental Records** (e.g., learning rate schedule, loss curves, validation performance changes, calculation of validation metrics)

- **Testing Results**: For the selected tooth IDs, each ID needs to be tested on at least 5 images, and the visualization results should be used as verification of the result accuracy. (The appendix includes code for visualizing test results.) Meanwhile, you need to attach the test image name and target id in result.

- **Submit a PDF report**

Dataset



The dataset images are mainly obtained by multi-view rendering of scanned 3D tooth models.Each view contains an RGB image and a mask image. The mask records the upper teeth, lower teeth, and one specific tooth segmentation label, which can be used for training and validating the segmentation network.The dataset is divided into training, validation, and test sets. The training and validation sets include ground-truth labels, while the provided test set is used to evaluate the segmentation accuracy of the trained network.The report must include complete testing on the test set, along with visualized results for grading by the teaching assistant.

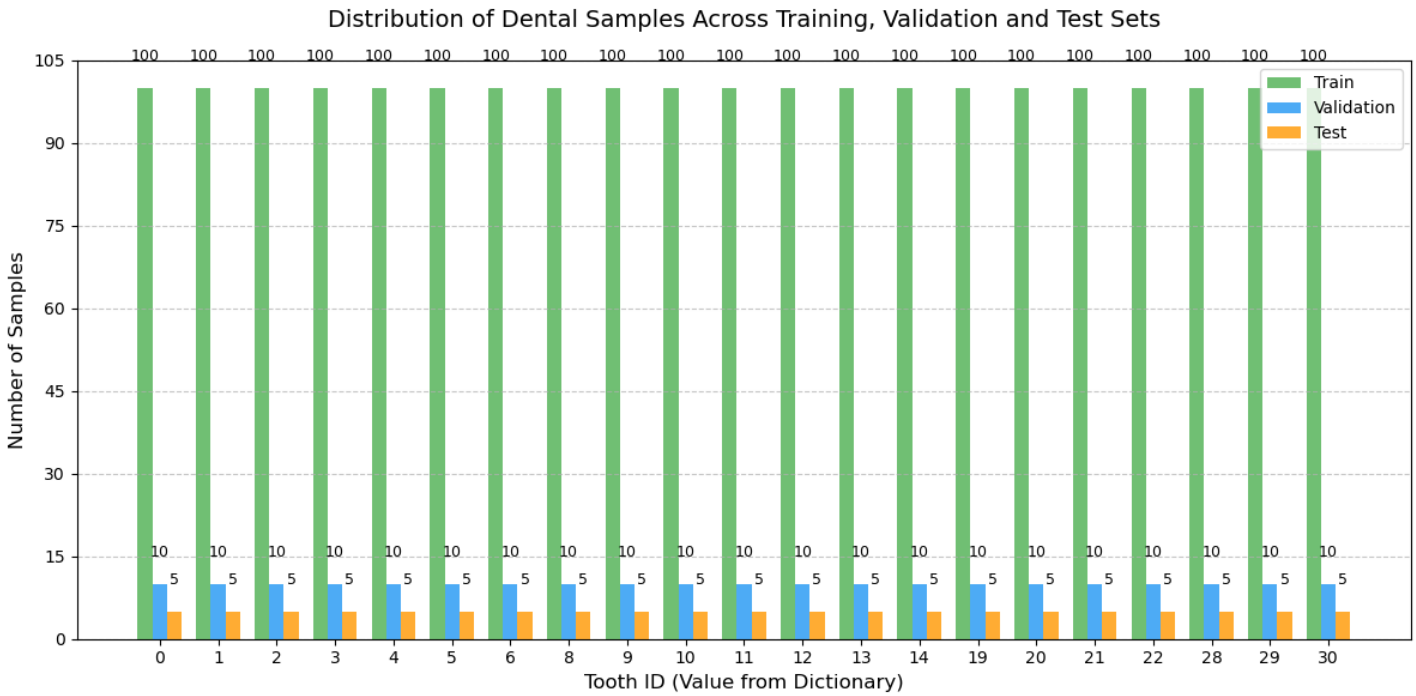
1. **Tooth and label ID**

tooth label id

```
{
    "11": 0,
    "12": 1,
    "13": 2,
    "14": 3,
    "15": 4,
    "16": 5,
    "17": 6,
    "21": 8,
    "22": 9,
    "23": 10,
    "24": 11,
```

```
    "25": 12,
    "26": 13,
    "27": 14,
    "34": 19,
    "35": 20,
    "36": 21,
    "37": 22,
    "45": 28,
    "46": 29,
    "47": 30
} # "11": represent name string, and 0 is tooth id
```

2. Dataset Statistics



dataset dir structure

```
├── testset
│   ├── 0000
│   │   ├── 000_rgb.jpg
│   │   ├── 001_rgb.jpg
.....
└── trainset_valset
    ├── 0
    │   ├── train
    │   │   ├── 10236497920_11
    │   │   │   ├── 000_mask.jpg
    │   │   │   ├── 000_rgb.jpg
```

```
12      |   └── val
13      |       ├── 1008_11
14      |       |   ├── 000_mask.jpg
15      |       |   ├── 000_rgb.jpg
16
17  # 0 repsents tooth id
```

3. **Mask image read**

parse mask image

```python
import cv2
import numpy as np
def read_mask(mask_path):
    # value:1 upper
    # value:2 lower
    # value:3 a tooth
    mask_img = cv2.imread(mask_path,-1)
    mask_upper = np.zeros_like(mask_img)
    mask_upper = np.where(mask_img==1,255,0)

    mask_lower = np.zeros_like(mask_img)
    mask_lower = np.where(mask_img==2,255,0)

    mask_tooth = np.zeros_like(mask_img)
    mask_tooth = np.where(mask_img==3,255,0)

    return mask_upper,mask_lower,mask_tooth
```

4. Dataset link

https://drive.google.com/file/d/1_KteV6HHH9oJO_NVStTw3-E0xAs8iAaN/view?usp=drive_link

Appendix:

- **Mean Pixel Accuracy (mPA)**

$$mPA = \frac{1}{k}\sum_i \frac{n_{ii}}{\sum_j n_{ij}}$$

Among them, *nij* denotes the number of pixels that belong to the true class *i* but are predicted as class *j*, and *nii* represents the number of pixels that are correctly predicted.

- **Intersection over Union (IoU)**

$$IoU_i = \frac{n_{ii}}{\sum_j n_{ij} + \sum_j n_{ji} - n_{ii}}$$

- **Mean IoU (mIoU)**

$$mIoU = \frac{1}{k} \sum_i IoU_i$$

scoring criteria:

Algorithm design	Report completeness	Testing result
30%	50%	20%

Mask visualization

mask vis

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import cv2

def generate_32_rgb_colors(cmap_name='hsv'):

    num_colors = 32
    samples = np.linspace(0, 1, num_colors, endpoint=False)

    cmap = cm.get_cmap(cmap_name, num_colors)
    rgba_colors = cmap(samples)
    rgb_colors = (rgba_colors[:, :3] * 255).astype(np.uint8)
    return rgb_colors

COLOR_32=generate_32_rgb_colors()

```

```python
def show_anns(image,input_mask, tooth_id=0,borders=True):
    '''
    image: is rgb image, for visualization [h,w,3]
    input_mask: is to visulaize mask , is from testing result.  [h,w]
    tooth_id: the target tooth id for this mask.
    '''
    assert tooth_id >=0 and tooth_id<32, "check tooth id ,it should in [0,31]"
    assert image.shape[:2] == input_mask.shape, "image and mask shape should same."

    alpha=0.7
    color = np.array(COLOR_32[tooth_id], dtype=np.float32)

    img = image.copy()
    mask = input_mask > 2
    img[mask] = img[mask] * (1 - alpha) + color * alpha

    if borders and np.any(mask):
        mask_uint8 = (mask.astype(np.uint8) * 255)
        contours, _ = cv2.findContours(mask_uint8, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        contours = [cv2.approxPolyDP(contour, epsilon=0.01, closed=True) for contour in contours]
        contour_color = (0, 0, 0, 0.8)
        contour_bgr = (contour_color[2], contour_color[1], contour_color[0], contour_color[3])
        cv2.drawContours(img, contours, -1, contour_bgr, thickness=1, lineType=cv2.LINE_AA)
    ax = plt.gca()
    ax.set_autoscale_on(False)
    # ax.imshow(img)
    return img
```