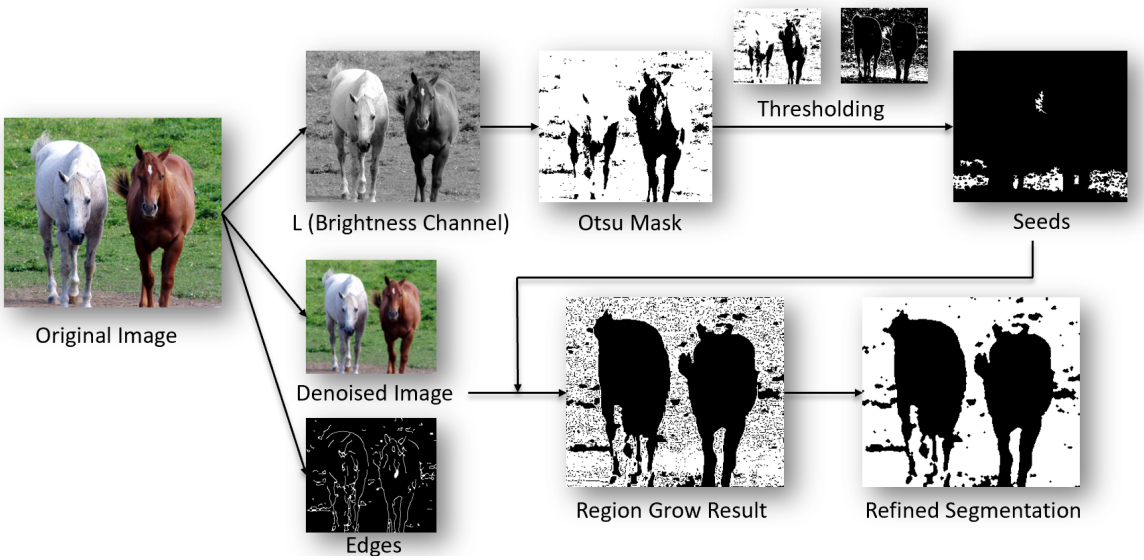


Figure 1: Segmentation Algorithm Overview

Course Design
Foreground Segmentation

Student: Yuanyuan Zhao, 2300013077, School of EECS, zhaoyuanyuan@stu.pku.edu.cn
Lecturer: Yisong Chen

Part 1: Algorithm Design Overview

As shown in Figure 1, we implemented a segmentation algorithm, comprising of (i) Enhanced Otsu Threshold Seed, (ii) Region Growing and (iii) Morphological Refinement.

(a) **Enhanced Otsu Threshold Mask**

Otsu is a segmentation method whose idea is to maximize the variance between Foreground and Background. Below is the math formula.

(i) Otsu Thresholding

$$\omega_0 = \sum_{i=0}^{k} p_i \quad \text{(Background Probability)}$$

$$\omega_1 = \sum_{i=k+1}^{L-1} p_i = 1 - \omega_0 \quad \text{(Foreground Probability)}$$

$$\mu_0 = \frac{\sum_{i=0}^{k} i \cdot p_i}{\omega_0} \quad \text{(Background Mean)}$$

$$\mu_1 = \frac{\sum_{i=k+1}^{L-1} i \cdot p_i}{\omega_1} \quad \text{(Foreground Mean)}$$

$$\mu_T = \sum_{i=0}^{L-1} i \cdot p_i \quad \text{(Global Mean)}$$

$$\sigma_B^2(k) = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 \text{(Variance)}$$

$$k^* = \arg\max_{0 \le k \le L-1} \sigma_B^2(k) \text{(Otsu Threshold)}$$

(ii) Adjust Coefficient

We adjust the coefficient of the threshold to make the threshold more adaptive in high-variance region.

```
local_var = cv2.boxFilter(L**2, -1, (window_size, window_size)) -
    cv2.boxFilter(L, -1, (window_size, window_size))**2
T_local = np.where(local_var > 100, T_global * 1.2, T_global)
```

(iii) Color Filter, Morphology Refinement and Connected Region Filter

Methods above are employed to further reduce false seeds.

```
delta_E = np.sqrt((A - mean_A)**2 + (B - mean_B)**2)
seeds = cv2.erode(seeds.astype(np.uint8), kernel)
density = np.sum(mask) / (np.prod(mask.shape))
```

(b) **Region Growing**

We considered the following three factors when distinguishing whether two pixels are in the same region, i.e. foreground or background.

(i) Color

```
seed_color = image_lab[y, x]
color_dist = np.linalg.norm(color - mean_color)
```

(ii) Texture

```
from skimage.feature import local_binary_pattern
lbp = local_binary_pattern(image_gray, P=8, R=1, method='uniform')
texture_dist = abs(texture - mean_texture)
```

(iii) Contour

Bilateral Filter, Canny Edge Detection and Morphology Refinement is employed to detect contours. When the seeds grow to the edge, the growing process stops.

(c) **Morphological Refinement**

(i) Erosion and Dilation

Erosion: Shrinks objects by removing boundary pixels

Dilation: Expands objects by adding boundary pixels

$$A \ominus B = \{z \mid B_z \subseteq A\} \text{(Erosion)}$$

$$A \oplus B = \{z \mid (\hat{B})_z \cap A \neq \emptyset\} \text{(Dilation)}$$

(ii) Opening and Closing Operators
Opening: Removes small objects and thin protrusions
Closing: Fills small holes and gaps

$$A \circ B = (A \ominus B) \oplus B \text{(Opening)}$$
$$A \bullet B = (A \oplus B) \ominus B \text{(Closing)}$$

(iii) Fill in small holes and Remove isolated noise

```
closed = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel, iterations=1)
opened = cv2.morphologyEx(closed, cv2.MORPH_OPEN, kernel, iterations=1)
```

(a) Damaged Image

(b) Seed Mask

(c) Region Grow Result

(d) Result Image

Figure 2: 1097 Result

Part 2: Result Analysis

(a) **Result Images**

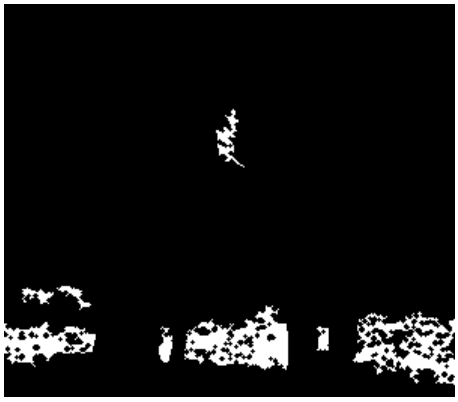
Final and middle results are shown in Figure 2, 3, 4, 5, which are under identical parameters. The results still have imperfections, but can be said to have achieved satisfactory segmentation using traditional methods without any deep learning strategy.

However, to achieve better performance, parameters (e.g. color_threshold, text_threshold) need to be adjusted with regard to different image features. In Figure 5, obviously the morphology refinement parameter needs to be adjusted, and in Figure 4 it is more than suitable.

(b) **Computational Time**

As shown in Table 1, compared to the BPP algorithm, our segmentation algorithm consumes more time, since it does not employ any multiprocessing or parallel technique and we emphasize the idea of designing rather than the efficiency of the algorithm.

(a) Damaged Image

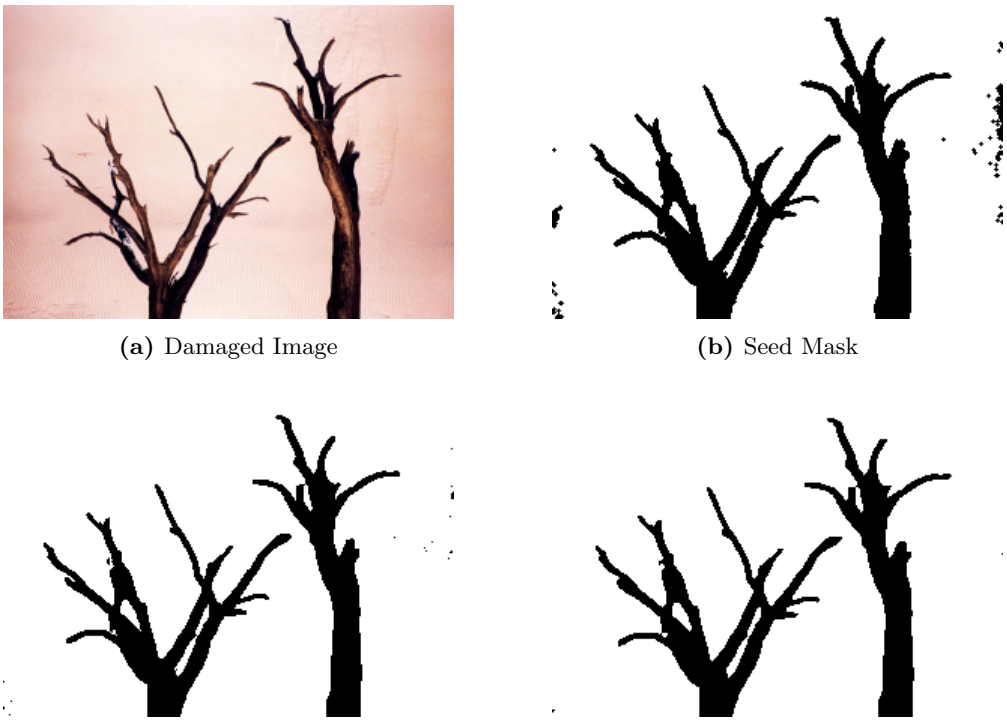


(b) Seed Mask

(c) Region Grow Result

(d) Result Image

Figure 3: 2009 Result



(a) Damaged Image **(b)** Seed Mask

(c) Region Grow Result **(d)** Result Image

Figure 4: 2010 Result

Computational Time	Ours(s)	BPP number_iter(s)	BPP iter_winner(s)
mean	482.31	2.23	1.63
std	573.35	0.36	0.23

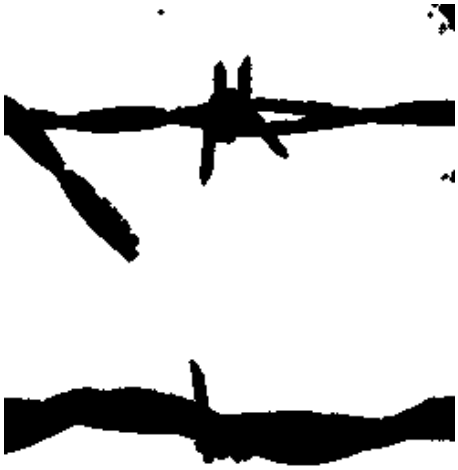
Table 1: Computational Time Comparison

Future work can be done to optimize the efficiency by turning for-loop into matrix operations, resorting to multiprocessing or CUDA and so on.

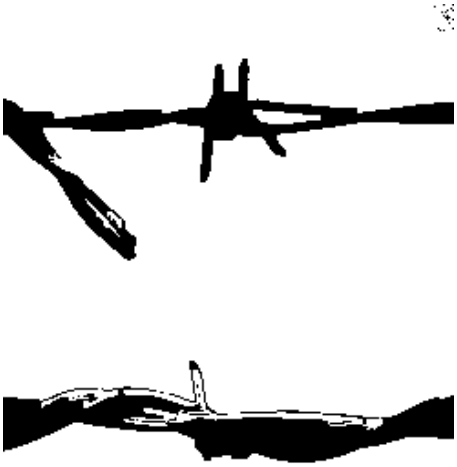
Part 3: Summary and Thoughts

I have tried a lot of strategies, either those taught in class or those found in some reference materials, to achieve better performance without using deep learning models. What I have learned most is to adjust (or add, remove) a specific module after visualizing the middle results and finding their problems, instead of wanting to reach perfection at the very first stage. In this process, I get the fun of carefully finding where the problem lies, thinking how to solve it and implementing my own idea, despite some inevitable bad results that make me frustrated sometimes.

(a) Damaged Image



(b) Seed Mask



(c) Region Grow Result

(d) Result Image

Figure 5: 2084 Result