**1.Пороговая сегментация**

Пример реализации алгоритма на Python

***# Загрузка изображения КТ***

***image = sitk.ReadImage("path\_to\_image.mhd")***

***# Выделение пороговых значений***

***lower\_threshold = 100***

***upper\_threshold = 200***

***# Сегментация***

***segmented\_image = sitk.BinaryThreshold(image, lowerThreshold=lower\_threshold,***

***upperThreshold=upper\_threshold,***

***insideValue=1, outsideValue=0)***

***# Сохранение результата***

***sitk.WriteImage(segmented\_image, "path\_to\_segmented\_image.mhd")***

**2.Сегментации региональным наращиванием**

**def region\_growing(image, seed\_point, threshold):**

**"""**

**Сегментация изображения реиональным наращиванием.**

**Args:**

**image: Входное изображение.**

**seed\_point: Начальная точка сегментации.**

**threshold: Пороговое значение для сходства пикселей.**

**Returns:**

**Маска сегментированного региона.**

**"""**

**# Получение размеров изображения**

**rows, cols = image.shape[:2]**

**# Создание маски для сегментированного региона**

**mask = np.zeros\_like(image, dtype=np.uint8)**

**# Инициализация очереди с начальной точкой**

**queue = [seed\_point]**

**# Цикл пока очередь не пуста**

**while queue:**

**# Извлечение текущей точки из очереди**

**current\_point = queue.pop(0)**

**# Проверка, была ли эта точка уже посещена**

**if mask[current\_point[1], current\_point[0]] == 0:**

**# Пометка точки как посещенной**

**mask[current\_point[1], current\_point[0]] = 255**

**# Получение пикселей вокруг текущей точки**

**neighbors = [(current\_point[0] - 1, current\_point[1]),**

**(current\_point[0] + 1, current\_point[1]),**

**(current\_point[0], current\_point[1] - 1),**

**(current\_point[0], current\_point[1] + 1)]**

**# Итерация по соседям**

**for neighbor in neighbors:**

**# Проверка, находится ли сосед в пределах изображения**

**if 0 <= neighbor[0] < cols and 0 <= neighbor[1] < rows:**

**# Проверка, похож ли сосед на текущую точку**

**if abs(image[neighbor[1], neighbor[0]] - image[current\_point[1], current\_point[0]]) <= threshold:**

**# Добавление соседа в очередь**

**queue.append(neighbor)**

**return mask**

**3. Метод сегментации на основе градиента**

**Пример:**

***import cv2***

***import numpy as np***

***# Загружаем изображение***

***image = cv2.imread("image.jpg", cv2.IMREAD\_GRAYSCALE)***

***# Вычисляем градиент с помощью оператора Sobel***

***grad\_x = cv2.Sobel(image, cv2.CV\_64F, 1, 0, ksize=5)***

***grad\_y = cv2.Sobel(image, cv2.CV\_64F, 0, 1, ksize=5)***

***# Вычисляем величину градиента***

***gradient = np.sqrt(grad\_x\*\*2 + grad\_y\*\*2)***

***# Применяем пороговую обработку для выделения границ***

***threshold = 100***

***edges = cv2.threshold(gradient, threshold, 255, cv2.THRESH\_BINARY)[1]***

***# Отображаем результат***

***cv2.imshow("Edges", edges)***

***cv2.waitKey(0)***

***cv2.destroyAllWindows()***

***4.*** Метод сегментации на основе активных контуров

**Пример:**

***import cv2***

***import numpy as np***

***# Загружаем изображение***

***image = cv2.imread("image.jpg", cv2.IMREAD\_GRAYSCALE)***

***# Инициализируем контур***

***contour = np.array([***

***[100, 100],***

***[150, 100],***

***[150, 150],***

***[100, 150],***

***], dtype=np.int32)***

***# Вычисляем градиент изображения***

***grad\_x = cv2.Sobel(image, cv2.CV\_64F, 1, 0, ksize=5)***

***grad\_y = cv2.Sobel(image, cv2.CV\_64F, 0, 1, ksize=5)***

***gradient = np.sqrt(grad\_x\*\*2 + grad\_y\*\*2)***

***# Создаем объект Snake***

***snake = cv2.ximgproc.createActiveContour()***

***# Устанавливаем параметры Snake***

***alpha = 0.015 # Сглаживание***

***beta = 10 # Жесткость***

***gamma = 0 # Внешние силы***

***# Устанавливаем градиент изображения в качестве внешней силы***

***snake.setGradientImage(gradient)***

***# Устанавливаем начальный контур***

***snake.setInitContour(contour)***

***# Запускаем алгоритм Snake***

***contour = snake.run(image, alpha, beta, gamma)***

***# Отображаем результат***

***cv2.drawContours(image, [contour], -1, (0, 255, 0), 2)***

***cv2.imshow("Snake", image)***

***cv2.waitKey(0)***

***cv2.destroyAllWindows()***

**5. Сегментация по водоразделу**

**Пример:**

***import skimage.morphology as morphology***

***from skimage import data***

***# Загружаем изображение***

***image = data.camera()***

***# Преобразуем изображение в карту высот***

***gradient = morphology.gradient(image)***

***# Выполняем сегментацию по водоразделу***

***segmentation = morphology.watershed(gradient, markers)***

***# Отображаем результат***

***plt.imshow(segmentation, cmap=plt.cm.gray)***

***plt.show()***

6. **Методы машинного обучения**

**Реализация на Python:**

***import tensorflow as tf***

***from tensorflow.keras import layers***

**# Определение модели U-Net**

***def build\_unet(input\_shape):***

***inputs = tf.keras.Input(shape=input\_shape)***

**# Свёрточный блок**

***conv1 = layers.Conv2D(64, 3, activation='relu', padding='same')(inputs)***

***conv1 = layers.Conv2D(64, 3, activation='relu', padding='same')(conv1)***

***pool1 = layers.MaxPooling2D(pool\_size=(2, 2))(conv1)***

***conv2 = layers.Conv2D(128, 3, activation='relu', padding='same')(pool1)***

***conv2 = layers.Conv2D(128, 3, activation='relu', padding='same')(conv2)***

***pool2 = layers.MaxPooling2D(pool\_size=(2, 2))(conv2)***

***conv3 = layers.Conv2D(256, 3, activation='relu', padding='same')(pool2)***

***conv3 = layers.Conv2D(256, 3, activation='relu', padding='same')(conv3)***

***pool3 = layers.MaxPooling2D(pool\_size=(2, 2))(conv3)***

***conv4 = layers.Conv2D(512, 3, activation='relu', padding='same')(pool3)***

***conv4 = layers.Conv2D(512, 3, activation='relu', padding='same')(conv4)***

***pool4 = layers.MaxPooling2D(pool\_size=(2, 2))(conv4)***

**# *Деконволюционный блок***

***up5 = layers.UpSampling2D(size=(2, 2))(pool4)***

***conv5 = layers.Conv2D(256, 3, activation='relu', padding='same')(up5)***

***conv5 = layers.Conv2D(256, 3, activation='relu', padding='same')(conv5)***

***up6 = layers.UpSampling2D(size=(2, 2))(conv5)***

***conv6 = layers.Conv2D(128, 3, activation='relu', padding='same')(up6)***

***conv6 = layers.Conv2D(128, 3, activation='relu', padding='same')(conv6)***

***up7 = layers.UpSampling2D(size=(2, 2))(conv6)***

***conv7 = layers.Conv2D(64, 3, activation='relu', padding='same')(up7)***

***conv7 = layers.Conv2D(64, 3, activation='relu', padding='same')(conv7)***

***outputs = layers.Conv2D(1, 1, activation='sigmoid')(conv7)***

***return tf.keras.Model(inputs, outputs)***

***# Загрузка данных и предобработка***

***# ...***

***# Создание модели***

***model = build\_unet(input\_shape=(256, 256, 1))***

**# *Компиляция модели***

***model.compile(***

***optimizer='adam',***

***loss='binary\_crossentropy',***

***metrics=['accuracy']***

***)***

***# Обучение модели***

***model.fit(***

***train\_images, train\_masks,***

***epochs=10,***

***validation\_data=(val\_images, val\_masks)***

***)***

***# Предсказание сегментации***

***predictions = model.predict(test\_images)***

***# Постобработка результатов***