HCC745 Signal und Bildverarbeitung – WS 2022
Übungsabgabe 4

Lisa-Marie Moser und Caroline Wagner

4. Februar 2023

Zusammenfassung

4.1 Region Growing

4.1.1 a)

Implementieren Sie einen Region Growing Algorithmus mit ImageJ, der sowohl ein Intensitätsintervall [minI;maxI] als auch eine Marker-Position (z.B. Rechtecks-Annotation) als Benutzereingabe erwartet. Als Nachbarschafts-Adjazenz erlauben Sie sowohl N4 als auch N8.. Inwieweit unterscheiden sich die Ergebnisse bei den beiden Adjazenzen? weisen Sie durch Tests nach.

Für das Region Growing muss zuerst sicher gestellt werden, das auch Startpunkte für die Suche nach Objekten mitgegeben erden. Dies wird in der Setup Methode durch den return Parameter ROI_REQUIRED sichergestellt. Mit dem Multi-Point Tool im ImageJ können dann verschiedenen Punkte angegeben werden.

```
1         public int setup(String arg, ImagePlus imp) {
2         if (arg.equals("about")) {
3             showAbout();
4             return DONE;
5         }
6         imPlus = imp;
7         return DOES_8G + ROI_REQUIRED;
8     } //setup
9
```

Des Weiteren werden als Eingabe der minimale und maximale Threshhold erwartet. Dies Eingabe wurde als GenericDialog umgesetzt. Die Werte können durch Slider eingegeben werden.

Während des Region Growings werden alle Hintergrund, die sich außerhalb des Threhholds befinden auf Schwarz, also 0 gesetzt und alle Werte die zu dem Object

gehören werden auf Weiß, also 255 gesetzt. Unprocessed, also -1 bedeutet, dass die Pixel noch nicht besucht wurden.

```
1 public static int BG_VAL = 0;  //background value ==> not the target objects
2 public static int FG_VAL = 255; //segmented object
3 public static int UNPROCESSED = -1; //not processed so far
```

Für das Region Growing selbst werden zuerst alle Pixel in unserem Ausgangsbild auf -1 (unprocessed) gesetzt. Als nächstes wird ein Stack angelegt indem alle Startpunkte gespeichert werden, die einerseits im Ausgangsbild noch unprocessed sind und an deren Stelle im Eingangsbildes sich der Wert innerhalb des festgelegten Threshholds befinden. Befinden sich die Punkte innerhalb des Treshholdes werden sie im Ausgangsbild als Vordergrund klassifiziert und bekommen den Wert 255. Sind sie außerhalb werden sie als Background klassifiziert und bekommen den Wert 0.

Von den Punkten ausgehend wird nun jeder Nachbar ebenfalls durchsucht, ob sich dieser noch im Bild selbst befindet, ob dieser noch unprocessed ist und sich innerhalb des Treshholdes befindet. Erfüllt ein Nachbarpunkt die Bedienungen wird er ebenfalls in den Stack gespeichert und als nächstes werden seine Nachbar untersucht. Dies wird solange wiederholt bis der Stack leer ist.

Der einzige Unterschied zwischen N8 und N4 ist, das bei N8 auch die diagobalen Nachbarn berücksichtige werden und bei N4 nur die vier unmittelbar daneben gelegenen.

Für die 8 Nachbarn wird über einen Offset von -1 bis 1 auf der x-Achse und y-Achse iteriert. Für die 4 Nachbarn wurde ein Offset Array erstell, das die offset Positionen der 4 nachbaren beinhaltet, wo über die einzelnen Inhalte iteriert wird. Damit die Methode zur Segmentierung auf eine Seite passt wurde die N4 Methode nicht angezeigt, die kann im Code nachgesehen werden. Zur veranschaulichung des Unterschiedes wurde allerdings folgender Code Teil herausgenommen.

```
1 int[] xOffset = new int[]{-1, 0, 0, 1};
2 int[] yOffset = new int[]{0, -1, 1, 0};
3
4 for (int i = 0; i < xOffset.length; i++) {
5     int nbx = actPos.x + xOffset[i];
6     int nby = actPos.y + yOffset[i];
```

Zum Abschluss werden noch alle unprosessed Pixel auch auf 0 gesetzt da sie ebenfalls zum Background gehören.

```
1  public int[][] segmentImage(int[][] inDataArrInt, int threshMax, int threshMin, int
       width, int height, String n) {
2      int[][] segmentedImg = new int[width][height];
3      bgCount = 0;
4      fgCount = 0;
5      //all pixels are unprocessed
6      for (int x = 0; x < width; x++) {
7          for (int y = 0; y < height; y++) {
8              segmentedImg[x][y] = UNPROCESSED;
9          }
10     }
11     Stack<Point> procStack = new Stack<>();
12     // let's check all the seeds
13     for (Point currSeed : GetSeedPositions(this.imPlus)) {
14         if (segmentedImg[currSeed.x][currSeed.y] == UNPROCESSED) {
15             int actVal = inDataArrInt[currSeed.x][currSeed.y];
16             if (actVal >= threshMin && actVal <= threshMax) { //interval matching
                                       segmentedImg[currSeed.x][currSeed.y] = FG_VAL;
17                 fgCount++;
18                 procStack.push(currSeed);
19             } else {
20                 segmentedImg[currSeed.x][currSeed.y] = BG_VAL;
21                 bgCount++;
22             }
23         }
24     }
25     // let's use N8
26     if (n == "N8") {
27         while (!procStack.empty()) {
28             Point actPos = procStack.pop();
29             //let's check the neighbours
30             for (int xOffset = -1; xOffset <= 1; xOffset++) {
31                 for (int yOffset = -1; yOffset <= 1; yOffset++) {
32                     int nbx = actPos.x + xOffset;
33                     int nby = actPos.y + yOffset;
34
35                     //check if insight the image
36                     if (nbx >= 0 && nbx < width && nby >= 0 && nby < height) {
37                         if (segmentedImg[nbx][nby] == UNPROCESSED) {
38                             int nbVal = inDataArrInt[nbx][nby];
39                             if (nbVal >= threshMin && nbVal <= threshMax) {
                                segmentedImg[nbx][nby] = FG_VAL;
40                                 fgCount++;
41                                 procStack.push(new Point(nbx, nby));
42                             } else {
43                                 segmentedImg[nbx][nby] = BG_VAL;
44                                 bgCount++;
45                             }
46                         }
47                     }
48                 }
49             }
50         }
51     }
52     //clean-up. all UNPROCESSED
53     for (int x = 0; x < width; x++) {
54         for (int y = 0; y < height; y++) {
55             if (segmentedImg[x][y] == UNPROCESSED) {
56                 segmentedImg[x][y] = BG_VAL;
57             }
58         }
59     }
60     return segmentedImg;
61 }
```

```
1 int[][] segmentedImg4 = segmentImage(inDataArrInt, threshMax, threshMin, width,
      height, "N4");
2 int[][] segmentedImg8 = segmentImage(inDataArrInt, threshMax, threshMin, width,
      height, "N8");
```

Die Methode segmentImage(int[][] inDataArrInt, int threshMax, int threshMin, int width, int height, String n) wird einmal mit dem String N8 und einmal mit dem Strin N4 aufgerufen. Beide Bilder werden dann mit ImageJUtility.showNewImage() ausgegeben.

Der Unterschied der beiden Methoden N8 und N4 ist das N4 die Objekte deutlicher abgrenzt, allerdings kann es dazu kommen, dass in Randbereichen nicht alles vom Objekt erfasst wird. Ganz deutlich wird es bei den folgenden Bildern, hier wird z.B. beim Reis mit N8 ein Reiskorn erkannt das nicht ausgewählt wurde, da es diagonal ein Pixel hatte das ebenfalls dem Treshhold entspricht, was bei N4 nicht der Fall war, selbiges gilt bei dem Fingerprint, hier wurde die Mitte des Fingerprints bei N8 als Teil des ausgewählten Objektes erkannt und bei N4 nicht. Bei dem Bild Coins ist weniger Unterschied außer das bei der obersten Münze der oberste Rand bei N4 nicht als Teil des Objektes erkannt wurde und bei N8 hingegen schon. Bei dem Bild patternMorph sind in den großen Objekten keine Unterschiede zu sehen, hingegen bei den markierten 8ern. Hier kann bei N8 noch ungefähr abgeschätzt werden, dass es sich um Zahlen handelt, bei N4 sieht es allerdings aus wie Punkte

So kann man grob sagen, dass wenn zwei Objekte diagonal aneinander angrenzen diese mit N8 nicht voneinander unterschieden werden können, bei N4 hingegen schon.

Der ungefähr beste Treshhold für das Region Growing wurde aus dem jeweiligen Histogramm des Bildes abgelesen. Das oberste Bild zeigt immer die Punkte die ausgewählt wurden.

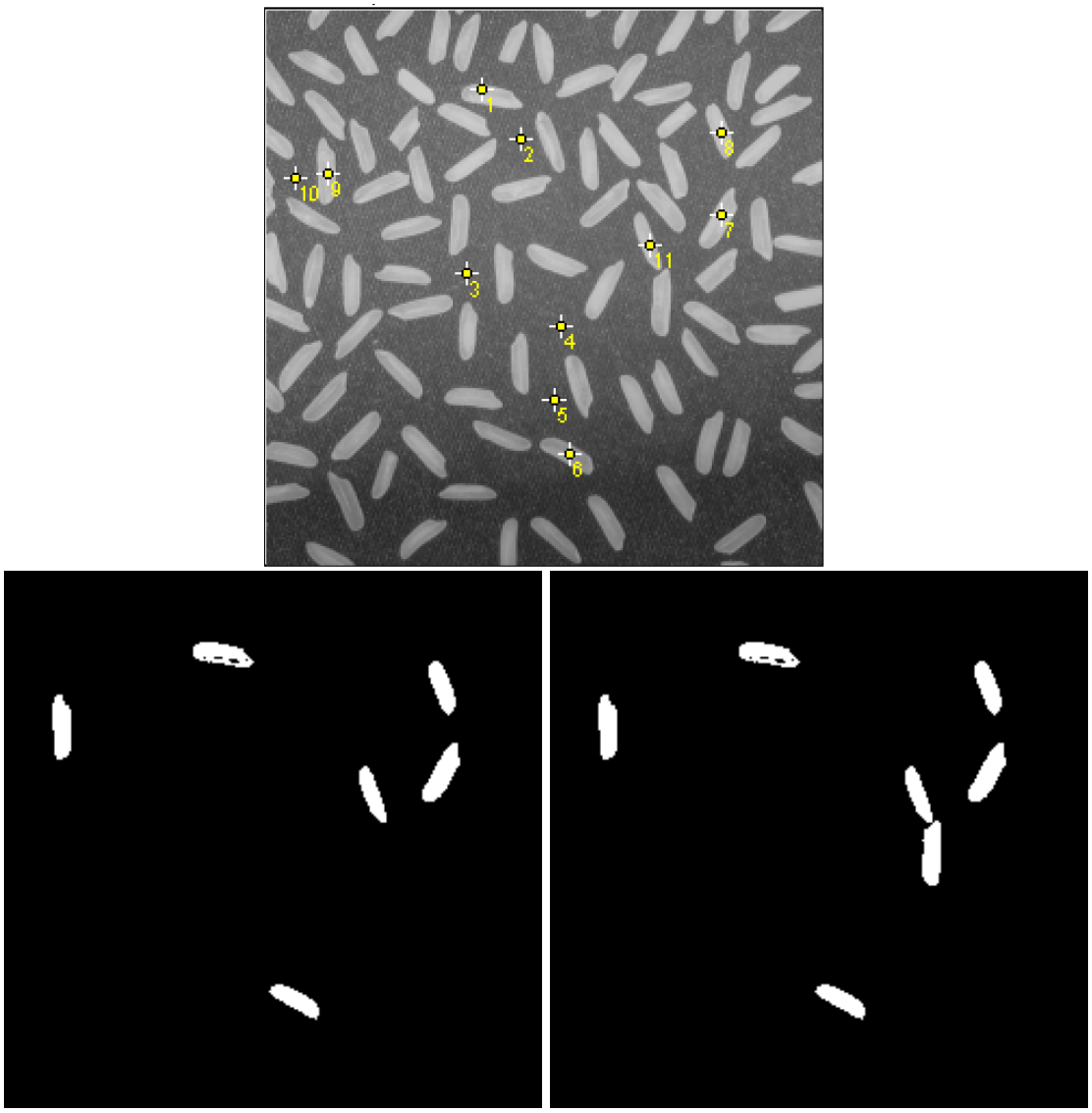

Abbildung 4.1: links N4, rechts N8, Treshhold: min=130, max=200

Abbildung 4.2: links N4, rechts N8, Treshhold: min=0, max=150

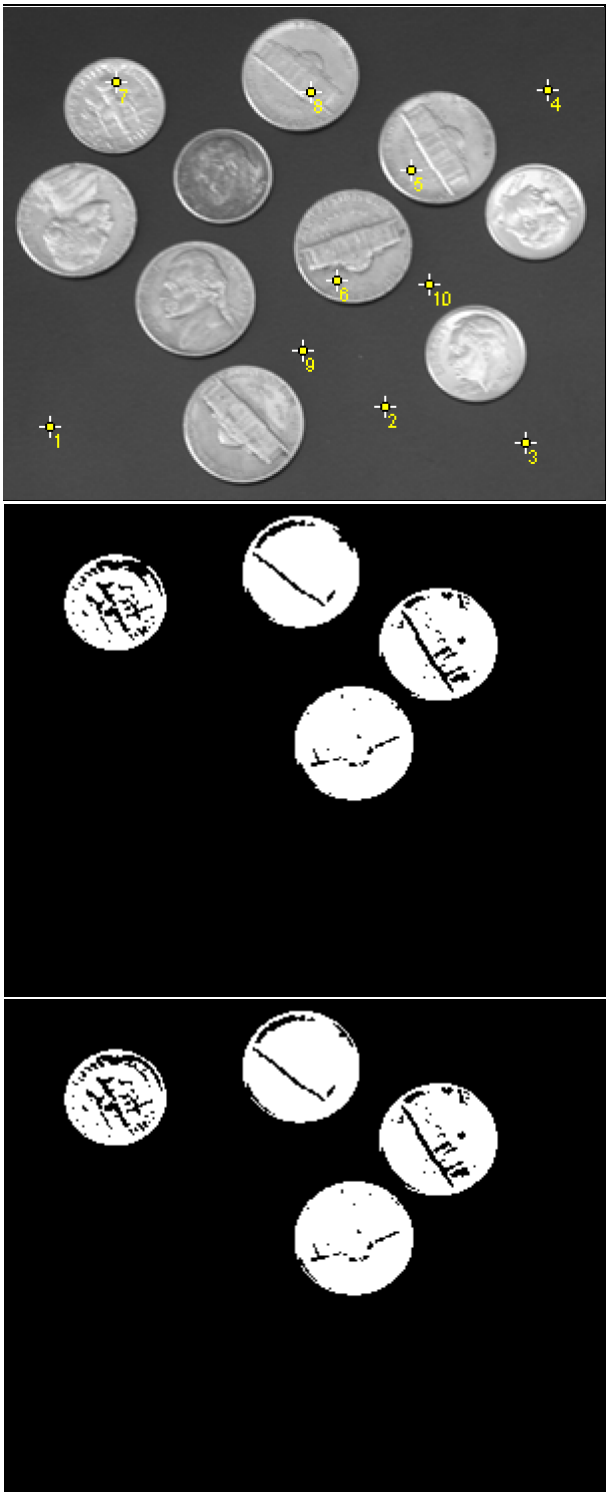

Abbildung 4.3: mitte N4, unten N8, Treshhold: min=100, max=200

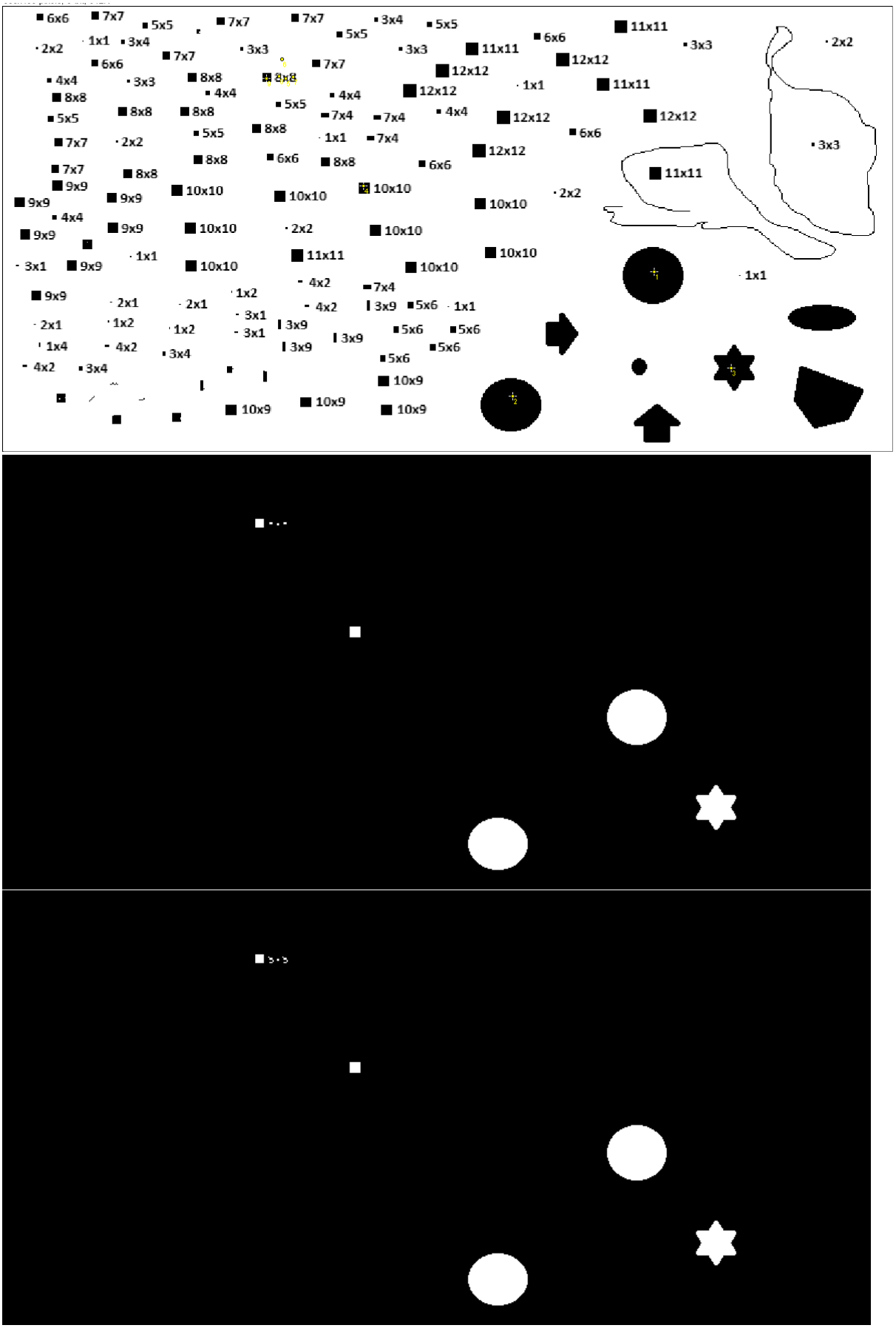

Abbildung 4.4: mitte N4, unten N8, Treshhold: min=0, max=10

4.2 Clustering-Algorithmen zur vollautomatischen Segmentierung von Farbbildern

Die Techniken der Bildsegmentierung lassen sich anhand der Bildeigenschaften in 3 Typen unterteilt werden: kantenbasierte Segmentierungsansätze Segmentierungsansätze, regionenbasierte Segmentierungsansätze und Schwellenwert-Segmentierungsansätze, sowie Ansätze mit tiefen neuronalen Netzen, hybride Ansätze und Clustering-Ansätze.

Segmentierung von Farbbildern:

Die Segmentierung von Bildern durch Clustering, sowie kMeans, bringt einige Probleme mit sich. So muss beispielsweise die Anzahl der Bildregionen a priori bekannt sein, und unterschiedliche Platzierung der Seeds (Anfangscluster) kann zu unterschiedlichen Segmentierungsergebnissen führen. Die meisten dieser Algorithmen könnten geringfügig verbessert werden, indem die Koordinaten des Bildes als Merkmale in den Clustering-Prozess einbezogen werden.

Eine neue Clustering-Methode beginnt mit einer automatischen Platzierung von Seeds unter Verwendung von Konturinformationen. Damit wird eine anfängliche Anzahl von Clustern im Bild bestimmt, die größer ist als die endgültige Anzahl von Regionen. Anschließend klassifiziert ein Clustering-Algorithmus die Pixel in Regionen. In dieser Phase gibt es zwei Einschränkungen: 1) Pixel mit einem hohen Gradienten werden nicht klassifiziert (Randregion), 2) Pixel, die weit vom Zentrum aller Cluster entfernt sind (Rauschpixel), werden nicht klassifiziert. Wenn das Clustering abgeschlossen ist, ist der Kern der Regionen segmentiert. Die Segmentierung der nicht klassifizierten Pixel wird anschließend unter Verwendung der Informationen des Kerns der Regionen sowie der Randinformationen durchgeführt. Der letzte Schritt besteht in der Zusammenführung der zuvor erhaltenen Regionen. Dieses Übersegmentierungsschema wird verwendet, um die Platzierung der Seeds in den Hauptregionen des Bildes zu gewährleisten.

Generell kann für die bessere Entscheidung über die Platzierung der ersten Seeds die Kanteninformationen verwendet werden. Um von jeder Region eine Stichprobe zu erhalten, die groß genug ist, um ihr Homogenitätsverhalten zu modellieren, müssen die ersten Seeds vollständig innerhalb der Regionen platziert werden. Mit Hilfe der Randinformationen kann die Positionen im Kern der Regionen extrahieren werden, indem nach Stellen weit weg von den Konturen gesucht wird. Die Hauptidee besteht darin, zu berechnen, wie stark sich die Konturen des Bildes auf die einzelnen Pixel des Bildes auswirken. Konturen, die sich in der Nähe eines Pixels befinden, beeinflussen dieses stärker als Konturen, die weit von ihm entfernt sind.

Eine neue Strategie für die Segmentierung von Clustern integriert, die Informationen über Regionen und Grenzen. Der Algorithmus verwendet Randinformationen, um auf unbeaufsichtigte Weise die Anzahl und den Ort der Seeds zu initialisieren, die als anfängliche zentrale Punkte des Clustering-Algorithmus verwendet werden. Das Clustering wird anschließend nur auf Pixel mit geringem Gradienten angewandt, um Grenzpixel nicht falsch zu klassifizieren. Anschließend werden die genauen Kanten bestimmt, indem die Nachbarschaft der Pixel mit den Merkmalen der angrenzenden Regionen verglichen wird. Schließlich ist ein Zusammenführungsschritt erforderlich, um benachbarte Regionen zu verschmelzen. Im Vergleich mit klassischen Clustering-Algorithmen wurde die Wirksamkeit des Algorithmus bestätigt.

Quellen: T. Raja and R. V. Babu, „A survey of image segmentation techniques“

B. Liu and S. Yang, „Color image segmentation: Advances and prospects“

A. Oliver, X. Munoz, J. Batlle, L. Pacheco and J. Freixenet, „Improving Clustering Algorithms for Image Segmentation using Contour and Region Information,“ 2006 IEEE International Conference on Automation, Quality and Testing, Robotics, Cluj-Napoca, Romania, 2006

G. B. Coleman and H. C. Andrews, „Image segmentation by clustering,“ Xu, D., Tian, Y. A Comprehensive Survey of Clustering Algorithms. Ann. Data. Sci. 2, 165–193 (2015)

Thilagamani, S., and N. Shanthi. „A survey on image segmentation through clustering.“ International Journal of Research and Reviews in Information Sciences 1.1 (2011): 14-17.

4.3 Semi-automatische Intensitätsbasierende Algorithmen zur benutzerzentrierten Segmentierung

Unter semi-automatischer Segmentierung versteht man eine Methode der Signal- und Bildverarbeitung, bei der der Benutzer in den Prozess der automatischen Segmentierung einbezogen wird. Bei diesem Verfahren wird der Benutzer zur Identifizierung bestimmter Regionen oder Merkmale im Bild herangezogen, die dann als Ausgangspunkt für die automatische Segmentierung verwendet werden.

Intensitätsbasierte Algorithmen sind eine Art von Algorithmen, die in der medizinischen Bildgebung verwendet werden, um bestimmte Regionen im Bild zu identifizieren, die sich in Bezug auf die Pixelintensität unterscheiden.

Einige bekannte halbautomatische intensitätsbasierte Algorithmen für die benutzerzentrierte Segmentierung sind:

Aktive Konturen (auch bekannt als Snake-Algorithmus) - dieser Algorithmus verwendet eine vom Benutzer gezeichnete Anfangskontur, um die Region im Bild zu segmentieren.

Level-Set-Methoden - diese Methoden verwenden eine vom Benutzer gezeichnete Anfangskontur, um den Bereich im Bild zu segmentieren, indem sie die Kontur anhand der Pixelintensitäten verfolgen.

GrabCut-Algorithmus - er erfordert Benutzerinteraktion, um die Regionen im Bild zu identifizieren, die als Hintergrund oder Vordergrund betrachtet werden sollen.

Graph Cut Algorithmus - er erfordert eine Benutzerinteraktion, um die Regionen im Bild zu identifizieren, die als Hintergrund oder Vordergrund betrachtet werden sollen.

Ein Nachteil bei semi-automatischer Segmentierung ist, dass weiterhin eine, wenn auch minimale, Benutzerinteraktion notwendig ist und diese auch Auswirkung auf die Ergebnisse und deren Qualität haben kann. Zusätzlich ist eine Eingabe durch einen Benutzer zeitaufwändig ist. Der Benutzer wird zwar meist nicht mehr benötigt, um ein zu segmentierendes Objekt vollständig abzugrenzen, sondern nur einen Bereich auszuwählen, dennoch ist eine vollautomatisierte Segmentierung nicht in gleicher Qualität möglich. Auch eine Segmentierung von Objekten mit unscharfen Rändern wie beispielsweise Tumore ist schwierig.

Semi-automatische Segmentierung ist bereits mit wenig Benutzeraktion möglich und bringt sehr gute Ergebnisse. An vollautomatisierten Segmentierungsmethoden wird zwar gearbeitet, die Qualität ist jedoch noch nicht auf demselben Standard wie jene mit (geringer) Benutzerinteraktion.

Quellen: A. Baâzaoui, W. Barhoumi, A. Ahmed, E. Zagrouba, Semi-Automated Segmentation of Single and Multiple Tumors in Liver CT Images Using Entropy-Based Fuzzy Region Growing

Wong, Damon, et al. „A semi-automated method for liver tumor segmentation based on 2D region growing with knowledge-based constraints.“ MICCAI workshop. Vol. 41. No. 43. 2008.

Baâzaoui, A., et al. „Semi-automated segmentation of single and multiple tumors in liver CT images using entropy-based fuzzy region growing.“ IRBM 38.2 (2017): 98-108.

Qin W, Wu Y, Li S, Chen Y, Yang Y, Liu X, Zheng H, Liang D, Hu Z. Automated segmentation of the left ventricle from MR cine imaging based on deep learning architecture. Biomed Phys Eng Express. 2020 Feb 18;6(2):025009. doi: 10.1088/2057-

1976/ab7363

Honti, Richard, Ján Erdélyi, and Alojz Kopáčik. „Semi-Automated Segmentation of Geometric Shapes from Point Clouds.“ Remote Sensing 14.18 (2022): 4591.

Gelaude, Frederik, Jos Vander Sloten, and Bert Lauwers. „Semi-automated segmentation and visualisation of outer bone cortex from medical images.“ Computer methods in biomechanics and biomedical engineering 9.1 (2006): 65-77.

Zusammenfassung und Anmerkungen

In dieser Übung wurde nur Aufgabe 4.1a) versucht und Aufgabe 4.5), die restlichen Aufgaben wurden nicht versucht! Für die Übung haben wir ungefähr 6 Stunden gebraucht, mit der Dokumentation.