Uniwersytet Zielonogórski
Instytut Sterowania i Systemów Informatycznych

Rozpoznawanie Obrazów

Semestr VII, Informatyka

mgr inż. Marcin Skobel

2021

Laboratorium nr 11: Segmentacja obrazu - metoda centroidów i metoda aktywnych konturów.

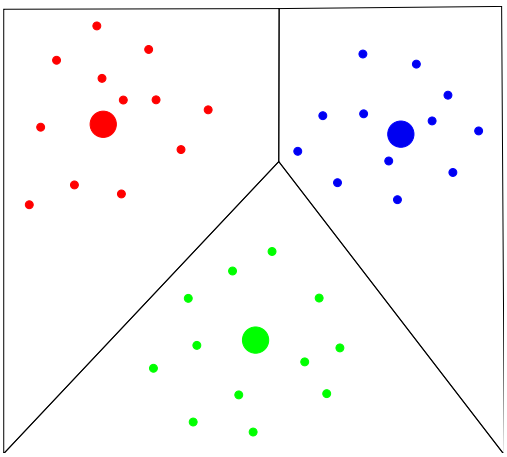
I. Zagadnienia teoretyczne

Wstęp

Bieżące laboratorium stanowi kontynuację rozległego tematu segmentacji obrazów. Segmentacja jak już wspomniano na poprzednich zajęciach służy do wyodrębniania homogenicznych obszarów obrazu cyfrowego. Dotychczas poznaliśmy metodę wododziałową oraz wstępne przetwarzanie obrazu do segmentacji za pomocą operacji rozplotu z odpowiednio dobranymi macierzami rozplotu. Na dzisiejszych zajęciach wykonamy segmentacje obrazów z użyciem klasteryzacji metodą centroidów (K-means) oraz omówimy działanie metody aktywnych konturów.

Algorytm centroidów (k-means, k-średnich)

Algorytm centroidów stanowi jedną z najpopularniejszych metod analizy skupień zwanych także klasteryzacją. Analiza skupień to proces polegający na grupowaniu obiektów o podobnych cechach do poszczególnych skupień (zbiorów obiektów wykazujących podobne cechy). Algorytm centroidów ma na celu uzyskać k różnych klastrów. Przyporządkowanie obiektów do klastrów odbywa się w sposób iteracyjny, aż do uzyskania kryterium zatrzymania, którym jest brak zmian położenia centroidów i obiektów w ostatniej iteracji. Algorytm dąży zatem do uzyskania optymalizacji wewnętrznej w grupach. Większość metod klasteryzacji wymaga podania wstępnej liczby klastrów w badanym zbiorze. Jeśli jednak przed wykonaniem eksperymentu liczba klastrów jest nieznana to można ją wyznaczyć z pomocą kryterium Daviesa-Bouldina lub kryterium Calinskiego-Harabasza. Większość przykładów stosowania analizy skupień odnosi się do przykładów dwuwymiarowych (Rys. 1) jednak nie ma żadnych przeszkód w stosowaniu klasteryzacji na obrazach cyfrowych, a nawet istnieją gotowe biblioteki w języku Python służące temu celowi.
Algorytm centroidów można rozpisać za pomocą następującego zbioru procedur:



Rysunek 1: Przykładowy obrazek po wykonaniu klasteryzacji k-means z trzema skupieniami

1. Wyznaczamy liczbę klastrów:
- Często bywa tak że wstępnie znamy liczbę klastrów w danym zbiorze lub możemy w przybliżeniu określić tą liczbę. Jeśli jednak nie jest ona znana istnieje kilka metod pozyskiwania optymalnej liczby klastrów np.: kryterium Daviesa-Bouldina, kryterium Calinskiego-Harabasza.
2. Inicjalizujemy wstępne położenie centroidów:
- Zadanie polega na wylosowaniu wstępnego położenia punktów centralnych skupień w przestrzeni. Można również ręcznie zdefiniować wstępne położenie centroidów. Ten krok algorytmu jest jednym ze słabszych punktów jego działania, przynajmniej w wersji losowego generowania wstępnego położenia centroidów.
3. Wyznaczenie odległości wszystkich obiektów do centroidów.
- Obliczenie odległości może się odbywać z użyciem dowolnej metryki jednak najczęściej stosowana jest metryka euklidesowa lub Czebyszewa.
4. Przypisanie obiektów do klastrów.
- W tym kroku każdy obiekt zostaje przypisany do tego skupienie do którego środka (centroidu) ma najbliżej.
5. Korekta położenia centroidów.
- Na tym etapie klastry zostały utworzone i czas na przesunięcie centroidu bliżej środka klastra. Środek w tym przypadku oznacza punkt którego współrzędne są średnią arytmetyczną współrzędnych położenia punktów znajdujących się w wyznaczonym klastrze.
6. Powtarzanie kroków 3,4,5 aż do uzyskania warunku zatrzymania.
- Warunek ten może stanowić z góry określona liczba iteracji albo moment w którym obiekty przestają się przenosić pomiędzy klastrami.

Metoda aktywnych konturów

Metoda aktywnych konturów jest jedną z metod segmentacji obiektów. Jest to metoda segmentacji krawędziowej w której zakłada się ciągłość krawędzi nawet jeśli takie nie są. Klasyczny model to parametryczny aktywny kontur czyli wąż (snake) oraz model nieco nowszy czyli geometryczny zwany też geodezyjnym (geodesic). Kolejne modyfikacje kompletnych konturów powstają pod wpływem przyłożonych do nich sił wewnętrznych i zewnętrznych natomiast

ewolucja aktywnego konturu polega na minimalizacji jego energii. Algorytm wąż jest zdefiniowany tak, że minimalizuje 3 energie - ciągłość, krzywiznę i gradient, które często są oznaczane jako: α, β, γ. Pierwsze dwa (razem nazywane energią wewnętrzną) są minimalizowane, gdy punkty (na krzywej) są coraz bliżej, tj. kurczą się. Jeśli się rozszerzają, zwiększa się energia, na co algorytm węża nie pozwala. Pierwotny algorytm zaproponowany w 1987 roku ma kilka problemów. Jednym z głównych problemów jest to, że na płaskich obszarach (gdzie gradient wynosi zero) algorytm nie jest w stanie wyznaczyć konturu. Zaproponowano kilka modyfikacji w celu rozwiązania tego problemu. Ciekawym rozwiązaniem jest tutaj - podejście siły balonu zaproponowana przez Cohena w 1989 roku. Siła balonu prowadzi kontur w nieinformacyjnych obszarach obrazu, tj. obszarach, w których gradient obrazu jest zbyt mały, aby przesunąć kontur w kierunku granicy. Wartość ujemna spowoduje zmniejszenie konturu, a wartość dodatnia spowoduje rozszerzenie konturu w tych obszarach. Ustawienie tej wartości na zero spowoduje wyłączenie siły balonu. Kolejnym ulepszeniem są - Węże morfologiczne, które używają operatorów morfologicznych (takich jak dylatacja lub erozja) na tablicy binarnej zamiast rozwiązywać równanie różniczkowe cząstkowe na tablicy zmiennoprzecinkowej, co jest standardowym podejściem dla aktywnych konturów. To sprawia, że węże morfologiczne są szybsze i bardziej stabilne numerycznie niż ich tradycyjny odpowiednik. Oprócz zadania segmentacji aktywne kontury są wykorzystywane do śledzenia ruchu.

II. Przykład praktyczny

Wstępna konfiguracja

Google Colaboratory podczas instalacji pakietu Histomicstk powoduje powstanie konfliktów pakietów. W tym celu instalujemy Histomicstk w poniższy sposób:

```
!pip install folium==0.2.1
!pip install histomicstk --find-links https://girder.github.io/large_image_wheels
!pip install pyyaml==5.4.1
```

Po instalacji musimy nacisnąć klawisz RESTART RUNTIME w celu przygotowania środowicka do pracy. Niestety również pakiet CV2 wymaga interwencji ponieważ wersja domyślna dla Colaboratory jest zbyt nowa i należy ją zredukować do wersji 4.1.2.30:

```
from scipy import ndimage as ndi
import matplotlib.pyplot as plt
import histomicstk as htk
!pip install opencv-python-headless==4.1.2.30
import cv2
import numpy as np
from skimage.morphology import disk
from skimage.segmentation import watershed
from skimage import data
from skimage.filters import rank
from skimage.util import img_as_ubyte
import scipy.ndimage
```

```
from google.colab import drive
drive.mount('/content/drive')
drive.mount('/content/drive')
```

Wczytanie pliku

```
#Wczytywanie obrazu
path = 'drive/My Drive/Colab Notebooks/Images/Test03.tif'
imInput = cv2.imread(path)
```

Rozplot (dekonwolucja) obrazu

```
# DEKONWOLUCJA
# Na poczatku nalezy zbudowac wektory do budowy macierzy dekonwolucji
# wektory sa wbudowane w biblioteke histomicstk
stain_color_map = htk.preprocessing.color_deconvolution.stain_color_map
print('stain_color_map:', stain_color_map, sep='\n')
# W kolejnym kroku tworzymy liste dzieki ktrorej wybierzemy odpowiednie wektory
# do zbudowania  macierzy dekonwolucji
stains = ['hematoxylin',  # nuclei stain
          'eosin',        # cytoplasm stain
          'null']         # set to null if input contains only two stains
# Przechodzimy do utworzenia macierzy dekonwolucji. Musimy wiedziec jakie
# barwniki zostaly uzyte przez lekarza
W = np.array([stain_color_map[st] for st in stains]).T
# Wykonanie dekonwolucji kolorow
imDec = htk.preprocessing.color_deconvolution.color_deconvolution(imInput, W)
# Wybor obrazu ze wzmocnieniem Hematoksylina
imHematox = imDec.Stains[:, :, 0]
plt.imshow(imHematox, cmap='gray')
plt.title('Hematoxylin', fontsize=16)
plt.show()
```

Segmentacja wododziałowa (wersja skimage)

```
# Progowanie metoda Otsu
ret2,th2 = cv2.threshold(imHematox,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
binary = 255-th2
plt.imshow(binary, cmap='gray')
plt.title('Otsu', fontsize=16)
plt.show()

#USUWANIE MALYCH OBIEKTOW
#odszukanie wszystkich pojedynczych elementow na obrazie
nb_comp, output, stats, ctr = cv2.connectedComponentsWithStats(binary, connectivity=8)
#connectedComponentswithStats zwraca kazdy oddzielny komponent z informacjami
#o kazdym z nich, takimi jak rozmiar. Nastepna czesc po prostu usuwa tlo, ktore
#rowniez jest uwazane za komponent, ale w wiekszosci przypadkow tego nie chcemy.
sizes = stats[1:, -1]; nb_comp = nb_comp - 1
#elementy mniejsze od tego progu beda usuwane (wielkosc w pikselach)
min_size = 1000
# deklaracja obrazu po usunieciu malych obiektow
DeleteSmall = np.zeros((output.shape), dtype='uint8')
```

```
#przeglad obiektow z pozostawieniem tylko najwiekszych
for i in range(0, nb_comp):
    if sizes[i] >= min_size:
        DeleteSmall[output == i + 1] = 255

plt.imshow(DeleteSmall, cmap='gray')
plt.title('After delete small objects', fontsize=16)
plt.show()

# WYPELNIENIE DZIUR
FillHolesStart = DeleteSmall/255
FillHoles = scipy.ndimage.binary_fill_holes(FillHolesStart).astype(int)
FillHoles = np.array(FillHoles*255, dtype='uint8')
plt.imshow(FillHoles, cmap='gray')
plt.title('After holes fill', fontsize=16)
plt.show()

#METODA WODODZIALOWA NA BAZIE GRADIENTU
# pozbywanie sie szumow
denoised = rank.median(imHematox, disk(2))
# wyszukiwanie obszarow ciaglych
markers = rank.gradient(denoised, disk(5)) < 10
markers = ndi.label(markers)[0]
# lokalny gradient sluzy do utrzymania cienkich krawedzi)
gradient = rank.gradient(denoised, disk(2))
# wykonaj wododzial
labels = watershed(gradient, markers)
#odcinamy obszary tla
labels[FillHoles==0]=0

plt.figure(figsize=(35, 30))
plt.imshow(labels, cmap = 'gist_ncar')
plt.title('Labels', fontsize=16)
plt.show()
```

Segmentacja algorytmem centroidów

```
# ANALIZA SKUPIEN - ALGORYTM CENTROIDOW
image = imHematox
pixel_values = image.reshape((-1, 3))
pixel_values = np.float32(pixel_values)
print(pixel_values.shape)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)
# deklaracja liczby klastrow
k = 3
_, labels, (centers) = cv2.kmeans(pixel_values, k, None, criteria, 10,
                                  cv2.KMEANS_RANDOM_CENTERS)
# konwersja do uint8
centers = np.uint8(centers)

# splaszczenie listy etykiet
labels = labels.flatten()
```

```
# konwersja wszystkich pikseli wedlug przynaleznosci do centroidu
segmented_image = centers[labels.flatten()]
# zmiana ksztaltu obrazu z powrotem do oryginalnego wymiaru
segmented_image = segmented_image.reshape(image.shape)
# wyniki
plt.imshow(segmented_image)
plt.show()
```

Segmentacja metodą aktywnych konturów

```
# METODA AKTYWNYCH KONTUROW
from skimage.segmentation import morphological_geodesic_active_contour
from skimage.segmentation import inverse_gaussian_gradient
from skimage.color import rgb2gray
from skimage.util import img_as_float
from PIL import Image, ImageDraw
# Tworzenie sztucznego obrazka na potrzeby testu dzialania aktywnego kontutu
im = Image.new('RGB', (250, 250), (128, 128, 128))
draw = ImageDraw.Draw(im)
draw.polygon(((50, 200), (200, 150), (150, 50), (100, 100)),
             fill=(255, 255, 0), outline=(0, 0, 0))
im = np.array(im)
im = rgb2gray(im)
im = img_as_float(im)
plt.imshow(im, cmap='gray')
# Teraz stworzmy funkcje, ktora pomoze nam przechowywac iteracje:
def store_evolution_in(lst):
    """Returns a callback function to store the evolution of the level sets in
    the given list.
    """

    def _store(x):
        lst.append(np.copy(x))

    return _store
# Ta metoda wymaga wstepnego przetworzenia obrazu w celu uwydatnienia konturow.
# Mozna to zrobic za pomoca funkcji inverse_gaussian_gradient, chociaz uzytkownik
# moze chciec zdefiniowac wlasna wersje. Jakosc segmentacji MorphGAC zalezy
# w duzym stopniu od tego etapu wstepnego przetwarzania.
gimage = inverse_gaussian_gradient(im)
# Ponizej okreslamy nasz punkt wyjscia - kwadrat.
init_ls = np.zeros(im.shape, dtype=np.int8)
init_ls[120:-100, 120:-100] = 1
# Lista z wynikami posrednimi do okreslenia ewolucji
evolution = []
callback = store_evolution_in(evolution)
# Teraz kluczowa linijka dla morphological_geodesic_active_contour
# z rozszerzeniem balonu jest ponizej:
ls = morphological_geodesic_active_contour(gimage, 50, init_ls,
                                           smoothing=1, balloon=1,
                                            threshold=0.7,
                                           iter_callback=callback)
```

```
# Na koniec przechodzimy do wyswietlania wynikow
fig, axes = plt.subplots(1, 2, figsize=(8, 8))
ax = axes.flatten()

ax[0].imshow(im, cmap="gray")
ax[0].set_axis_off()
ax[0].contour(ls, [0.5], colors='b')
ax[0].set_title("Morphological GAC segmentation", fontsize=12)

ax[1].imshow(ls, cmap="gray")
ax[1].set_axis_off()
contour = ax[1].contour(evolution[0], [0.5], colors='r')
contour.collections[0].set_label("Starting Contour")
contour = ax[1].contour(evolution[5], [0.5], colors='g')
contour.collections[0].set_label("Iteration 5")
contour = ax[1].contour(evolution[-1], [0.5], colors='b')
contour.collections[0].set_label("Last Iteration")
ax[1].legend(loc="upper right")
title = "Morphological GAC Curve evolution"
ax[1].set_title(title, fontsize=12)

plt.show()
```

III. Uwagi

Do wykonania dzisiejszego przykładu warto pobrać plik testowy dostępny pod adresem:
`http://staff.uz.zgora.pl/mskobel/Test03.tif`
Plik z przykładami można pobrać ze strony: `http://staff.uz.zgora.pl/mskobel/lab11.py`

IV. Lista zadań

1. Proszę wybrać dowolny obraz histopatologiczny z zajęć (laboratoryjnych lub projektowych) z zaznaczonymi ręczne jądrami komórkowymi lub przygotować ręcznie oznaczenie. Następnie przetworzyć ten obraz metodą wododziałową (należy eksperymentalnie dobrać najlepsze parametry) a następnie porównać liczbę obiektów po segmentacji manualnej oraz po segmentacji wododziałowej.