Федеральное агентство по образованию Российской Федерации
Государственное образовательное учреждение
высшего профессионального образования
Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе №2
«Обработка изображений с помощью бибиотеки OpenCV»

Выполнили:
студенты ф-та ИИТММ гр. 381908-1
Гордеев. В.В.
Шурыгина А.К.
Витулин И.А.
Проверила:
ассистент кафедры МОСТ, ИИТММ
Гетманская А.А.

Нижний Новгород
2021

Содержание

Введение

OpenCV — библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым исходным кодом. Реализована на языках программирования C/C++. Помимо этого разрабатываются версии для Python, Java, Ruby, Matlab, Lua и других ЯПВУ. Одной из основных оособенностей состоит в том, что все OpenCV массивы конвертируются в Numpy массивы. Это помогает совместному использованию библиотек также использующих Numpy, таких как SciPy и Matplotlib.

Ещё одним важным моментом является то, что библотека может свободно использоваться в академических и коммерческих целях, поскольку она распространяется в условиях лицензии BSD (что не может не радовать)

Постановка задачи

Имеется несколько изображений с листком дерева. С помощью функций и методов, реализованных в используемой библотеке, выедлить здоровую и повреждённую части листка на изображении. Разрешено использовать фильтры, уменьшающие шум и алгоритм водораздела (“watershared”)

Описание структур данных

- def NonLocalMeans(img, n) – изображение поступает на обработку нелокальным сглаживающим фильтром с последующим выводом результата
  - cv.fastNlMeansDenoisingColored() – функция нелокального сглаживающего фильтра
- def BilaterialFilter(img, n) – изображение поступает на обработку билатериальным фильтром с последующим выводом результата
  - cv.bilateralFilter() – функция билатериального фильтра
- def CalcOfDamageAndNonDamage() – изображение поступает на обработку, где выделяются здоровая и повреждённая части листа
  - cv.watershed() – функция алгоритма водораздела
- def MedianBlur() – изображение поступает на обработку медианным фильтром с последующим выводом результата
- def GaussianFilter() – изображение поступает на обработк фильтром Гаусса с последующим выводом результата

Описание алгоритмов

Фильтр "Non-Local Means"

Основная идея нелокального сглаживающего фильтра заключается в том, чтобы использовать всю информацию на изображении, а не только пиксели соседние с обрабатываемым. Если у нас есть несколько изображений одного и того же объекта с разным уровнем шума, то мы можем скомпоновать их в единую картинку без шума. Если эти объекты выглядят немного по-разному или частично перекрыты, то у нас всё равно есть избыточные данные, которыми можно воспользоваться. Нелокальный сглаживающий фильтр использует эту идею: находит похожие области изображения и применяет информацию из них для взаимного усреднения. Разумеется, если части изображения похожи, это не означает, что они принадлежат одним и тем же объектам. Но, как правило, приближение оказывается достаточно хорошим.

Билатериальный фильтр

Билатеральный фильтр – это фильтр нелинейного сглаживания изображения с сохранением его границ; расширение гауссового фильтра. Для размытия будут рассматриваться только пиксели со значениями интенсивности, подобными значению яркости центрального пикселя. Двусторонняя фильтрация также использует фильтр Гаусса в пространстве, но дополнительно учитывает еще один фильтр Гаусса, который является функцией разности пикселей

Алгоритм водораздела

Алгоритм водораздела основан на концепции визуализации изображения как топографической поверхности, где значения высокой интенсивности обозначают пики и холмы, а низкая интенсивность обозначает впадины. На такой пверхности можно определить три типа точек:

1) Точки локального минимума
2) Точки, в которые, если вы поместите каплю воды, эта капля наверняка упадет до единственного минимума.
3) Точки, в которых падение с одинаковой вероятностью упадет до более чем 1 такого минимума

Теперь множество точек определенного минимума,удовлетворяющих второму условию, называется водосборным бассейном или водоразделом этого минимума. А те, которые удовлетворяют третьему условию, называются линиями разделения или линиями водоразделов. Предположим, что в каждом локальном минимуме пробита дыра. Начинаем «заполнять» каждый локальный минимум (из этих отверстий) водой разного цвета (метки) с одинаковой скоростью. Через какое-то время вода из разных водосборных бассейнов или долин начнет переливаться. Чтобы предотвратить это, в местах слияния воды сооружаются дамбы или заграждения. Наступит этап, когда все вершины окажутся под водой, а видны только вершины плотин (преград). Эти границы плотины соответствуют линиям водоразделов. Но тут возникает проблема черезмерной сегментации из-за шума или других неровностей изображения. Для избежания этого были придуманы маркеры – набор пикселей, с которых начнется заливка. Другими словами – это пиксели, в которых мы уверены, что они принадлежат объектам, присутствующим на изображении. Обычно количество маркеров

равно количеству объектов (классов) + 1 (для фона). Эти маркеры могут быть либо явно определены пользователем, либо могут быть определены автоматически с помощью морфологических операторов или другими методами.

Заключение

После выполнении данной лабораторной работы мы познакомились с библиотекой OpenCV, а также смогли написать программу для обработки изображения с листком дерева и выделении на нём здоровых и повреждённых

Приложение

Код программы

```
import cv2 as cv
import sys
import numpy as np
from matplotlib import pyplot as plt

#Чтение изображений
img1 = cv.imread('1.jpg')
img2 = cv.imread('2.jpg')
img3 = cv.imread('3.jpg')
img4 = cv.imread('4.jpg')
img5 = cv.imread('5.jpg')
img6 = cv.imread('6.jpg')
img7 = cv.imread('7.jpg')
img8 = cv.imread('8.jpg')
img9 = cv.imread('9.jpg')
img10 = cv.imread('10.jpg')
img11 = cv.imread('11.jpg')
img12 = cv.imread('12.jpg')

i = 211

#Нелокальный сглаживающий фильт
def NonLocalMeans(img, n):
    b, g, r = cv.split(img)
    rgb_img = cv.merge([r, g, b])

    dst = cv.fastNlMeansDenoisingColored(img, None, 10, 10, 7, 21)
    b, g, r = cv.split(dst)  # g e t b , g , r
    rgb_dst = cv.merge([r, g, b])  # swi t c h i t t o rgb
    plt.subplot(n), plt.imshow(rgb_img)
    n += 1
    plt.subplot(n), plt.imshow(rgb_dst)
    plt.show()


NonLocalMeans(img1, i)
NonLocalMeans(img2, i)
NonLocalMeans(img3, i)
NonLocalMeans(img4, i)
NonLocalMeans(img5, i)
NonLocalMeans(img6, i)
NonLocalMeans(img7, i)
NonLocalMeans(img8, i)
NonLocalMeans(img9, i)
NonLocalMeans(img10, i)
NonLocalMeans(img11, i)
NonLocalMeans(img12, i)

#Билатериальный фильтр
def BilaterialFilter(img, n):
    b, g, r = cv.split(img)
    rgb_img = cv.merge([r, g, b])
    bilateral = cv.bilateralFilter(img, 40, 25, 75)
    plt.subplot(n), plt.imshow(rgb_img)
    n += 1
    plt.subplot(n), plt.imshow(bilateral)
    plt.show()
```

```
58
59
60      BilaterialFilter(img1, i)
61      BilaterialFilter(img2, i)
62      BilaterialFilter(img3, i)
63      BilaterialFilter(img4, i)
64      BilaterialFilter(img5, i)
65      BilaterialFilter(img6, i)
66      BilaterialFilter(img7, i)
67      BilaterialFilter(img8, i)
68      BilaterialFilter(img9, i)
69      BilaterialFilter(img10, i)
70      BilaterialFilter(img11, i)
71      BilaterialFilter(img12, i)
72
73      #Функция обнаружения повреждённой части листа
74      def CalcOfDamageAndNonDamage(image_name):
75          image = cv.imread(image_name)
76          kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (7, 7))
77          image_erode = cv.erode(image, kernel)
78          hsv_img = cv.cvtColor(image_erode, cv.COLOR_BGR2HSV)
79          markers = np.zeros((image.shape[0], image.shape[1]), dtype="int32")
80          markers[90: 140, 90: 140] = 255
81          markers[236: 255, 0: 20] = 1
82          markers[0: 20, 0: 20] = 1
83          markers[0: 20, 236: 255] = 1
84          markers[236: 255, 236: 255] = 1
85          leafs_area_bgr = cv.watershed(image_erode, markers)
86          healthy_part = cv.inRange(hsv_img, (36, 25, 25), (86, 255, 255))
87          ill_part = leafs_area_bgr - healthy_part
88          mask = np.zeros_like(image, np.uint8)
89          mask[leafs_area_bgr > 1] = (255, 0, 255)
90          mask[ill_part > 1] = (0, 0, 255)
91          return mask
92
93      #Медианный фильтр
94      def MedianBlur(img,n):
95          blur = cv.blur(img,(5,5))
96          plt.figure(figsize=(10,8),dpi=100)
97          plt.subplot (n), plt.imshow (img [:,:, :: - 1]), plt.title ('img')
98          plt.xticks([]), plt.yticks([])
99          plt.subplot (n), plt.imshow (blur [:,:, :: - 1]), plt.title ('result')
100         plt.xticks([]), plt.yticks([])
101         plt.show()
102
103     MedianBlur(img1, i)
104     MedianBlur(img2, i)
105     MedianBlur(img3, i)
106     MedianBlur(img4, i)
107     MedianBlur(img5, i)
108     MedianBlur(img6, i)
109     MedianBlur(img7, i)
110     MedianBlur(img8, i)
111     MedianBlur(img9, i)
112     MedianBlur(img10, i)
113     MedianBlur(img11, i)
114     MedianBlur(img12, i)
115
```

```
#Фильтр Гаусса
def GaussianFilter(img,n):
    gauss = cv.GaussianBlur(img, (10, 8),0)
    plt.figure(figsize=(11,6))
    plt.subplot(121), plt.imshow(img, cmap='gray'),plt.title('Original')
    plt.xticks([]), plt.yticks([])
    plt.subplot(122), plt.imshow(gauss, cmap='gray'),plt.title('Gaussian Filter')
    plt.xticks([]), plt.yticks([])
    plt.show()

GaussianFilter(img1,i)
GaussianFilter(img2,i)
GaussianFilter(img3,i)
GaussianFilter(img4,i)
GaussianFilter(img5,i)
GaussianFilter(img6,i)
GaussianFilter(img7,i)
GaussianFilter(img8,i)
GaussianFilter(img9,i)
GaussianFilter(img10,i)
GaussianFilter(img11,i)
GaussianFilter(img12,i)
```