



МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (МАИ)

Отчет о лабораторной работе №2

По дисциплине: **«Поиск, обработка и распознавание аудио, видео и графической информации»**

На тему **«Изучение и освоение методов классификации формы изображений»**

Выполнил:

Щербаков В.С. гр. М8О-110М-19

подпись, дата

Проверил:
проф.

Местецкий Леонид Моисеевич, д.ф.-м.н.,

подпись, дата

Москва, 2020

Содержание

1.	Постановка задачи.....	3
2.	Описание данных	4
3.	Описание метода решения	5
4.	Описание программной реализации.....	6
5.	Эксперименты	7
6.	Выводы.....	8
	Приложение: исходный код.....	9

1. Постановка задачи

Разработать и реализовать программу для классификации изображений ладоней, обеспечивающую:

- Ввод и отображение на экране изображений в формате TIF;
- Сегментацию изображений на основе точечных и пространственных преобразований;
- Генерацию признаков описаний формы ладоней на изображениях;
- Вычисление меры сходства ладоней;
- Кластеризацию изображений.

В качестве исходных данных прилагается набор из 99 цветных изображений ладоней разных людей, полученных с помощью сканера, в формате 489×684 с разрешением 72 dpi. Задача состоит в построении меры сходства изображений на основе выделения и анализа формы ладоней. Нужно разработать и реализовать алгоритм, входом которого является изображение, а выходом – описание признаков формы, попарные расстояния, кластеры изображений. Примеры входных изображений представлены на рисунках.

В качестве признакового описания формы предлагается построить «линию пальцев» - ломаную линию, соединяющую точки на кончиках пальцев (tips) с точками в основаниях пальцев (valleys). Пример такой линии представлен на рисунке. Длины 8 звеньев ломаной линии образуют 8-мерный вектор признаков формы ладони.

В задание входят задачи двух уровней сложности: Intermediate, Expert. Класс Intermediate:

1. Найти на изображении ладони точки в кончиках и основаниях пальцев.
 2. Визуализировать результат для экспертного контроля в виде картинки аналогичной приведенному выше рисунку.
- Класс Expert:
3. Найти для каждой ладони 3 наиболее похожих изображения и представить результат в виде таблицы «имя образца – имена ближайших соседей».
 4. Определить число людей, чьи ладони представлены в изображениях, и составить списки ладоней для каждого, т.е. провести кластеризацию изображений в виде таблицы «Персона № – имена изображений ладоней».

2. Описание данных

3. Описание метода решения

4. Описание программной реализации

Для решения данной задачи была разработана программа на языке python 3.7. Реализовано два варианта её использования:

1. Как консольное приложение
2. Как оконное приложение

Исходный код приведен в приложении

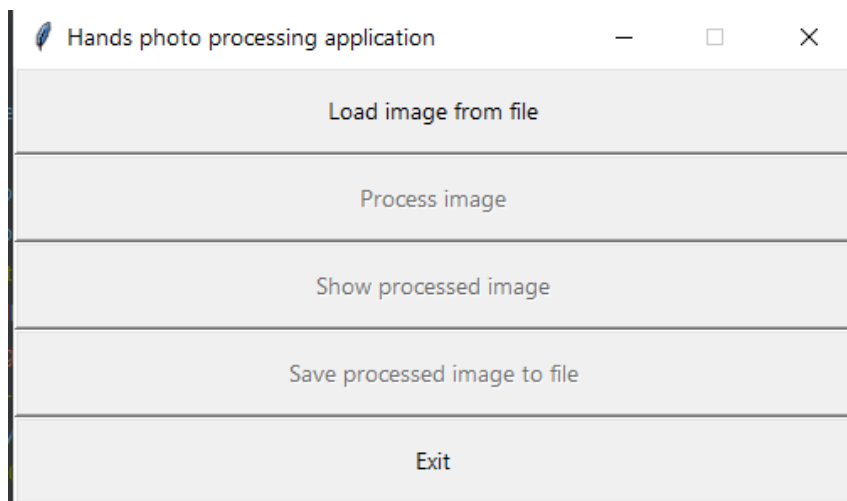


Рисунок 1 Скриншот работы программы

5. Эксперименты

В ходе написания программы отладка проводилась с использованием предложенных изображений. За счет того, что они имели достаточно различающийся вид, удалось добиться определенного уровня универсальности программы, что позволяет без изменения параметров конфигурации фильтров обрабатывать различные изображения, в том числе с сложным для распознавания фоном.

6. Выводы

В ходе работы была разработана программа на языке python 3.7 и освоены основополагающие методы обработки изображений и алгоритмы компьютерного зрения и их реализации для данного языка.

Приложение: Исходный код

```
#task_02.py

from image_processor import ImageProcessor
from cmd_pasrer import ImageProcessorArgs
from image_processing_app import ImageProcessingApp

if __name__ == "__main__":
    image_processor_args = ImageProcessorArgs()

    if image_processor_args.input_file_name is None:
        image_processor = ImageProcessor()
        app = ImageProcessingApp(image_processor)
        app.mainloop()
    else:
        input_file_name, output_file_name, approx_poly_dp_coefficient = image_processor_args.input_file_name, \
            image_processor_args.output_file_name, \
            image_processor_args.approx_poly_dp_coefficient
        image_processor = ImageProcessor(input_file_name, output_file_name, approx_poly_dp_coefficient)

        show_needed = image_processor_args.is_showing_picture_needed
        image_processor.load_image_from_file()
        image_processor.find_points()
        image_processor.save_image_to_file()
        if show_needed:
            image_processor.show_processed_image()
```

```
#image_processor.py

import numpy as np
import cv2
import os

class ImageProcessor:
    def __init__(self, input_file_name=None, output_file_name=None, coefficient=None):
        self._image_with_points = np.ndarray(shape=10)
        self._image_without_points = np.ndarray(shape=10)
        self._input_file_name = input_file_name
        self._approx_poly_dp_coefficient = 0.01 if coefficient is None else coefficient

    def find_points(self):
        img = self._image_without_points
        width = int(img.shape[1])
        height = int(img.shape[0])
        resize = cv2.resize(img, (width, height), interpolation=cv2.INTER_AREA)

        # Обесцвечиваем изображение
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
        lower_color = np.array([0, 35, 50])
        upper_color = np.array([30, 255, 255])

        mask = cv2.inRange(gray, lower_color, upper_color)

        gray = cv2.cvtColor(resize, cv2.COLOR_BGR2GRAY) # Обесцвечиваем изображение

        # kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2, 2))
        # hsv_d = cv2.dilate(mask, kernel)

        # cv2.imshow("Dilate", hsv_d)
```

```

# cv2.waitKey(0)

maxIntensity = 255.0 # depends on dtype of image data
x = np.arange(maxIntensity)

phi = 1
theta = 1

newImage0 = (maxIntensity / phi) * (mask / (maxIntensity / theta)) ** 0.5
newImage0 = np.array(newImage0, dtype="uint8")

blur = cv2.GaussianBlur(mask, (15, 15), 0)
ret, thresh1 = cv2.threshold(blur, 50, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
MB = cv2.medianBlur(blur, 19)

contours, hierarchy = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
drawing = np.zeros(img.shape, np.uint8)
max_area = 0

for i in range(len(contours)):
    cnt = contours[i]
    area = cv2.contourArea(cnt)
    if area > max_area:
        max_area = area
        ci = i
        cnt = contours[ci]
        hull = cv2.convexHull(cnt)
        moments = cv2.moments(cnt)
        if moments['m00'] != 0:
            cx = int(moments['m10'] / moments['m00']) # cx = M10/M00
            cy = int(moments['m01'] / moments['m00']) # cy = M01/M00

            central = (cx, cy)
            cv2.circle(img, central, 5, [0, 0, 255], 1)
            cv2.drawContours(drawing, [cnt], 0, (0, 255, 0), 1)
            cv2.drawContours(drawing, [hull], 0, (0, 0, 255), 1)
            cnt = cv2.approxPolyDP(cnt, self._approx_poly_dp_coefficient * cv2.arcLength(cnt, True), True)
            hull = cv2.convexHull(cnt, returnPoints=False)

            defects = cv2.convexityDefects(cnt, hull)
            for j in range(defects.shape[0]):
                s, e, f, d = defects[j, 0]
                start = tuple(cnt[s][0])
                end = tuple(cnt[e][0])
                far = tuple(cnt[f][0])
                cv2.circle(img, far, 3, [0, 0, 255], -1)
                cv2.circle(img, end, 3, [0, 0, 255], -1)
                cv2.circle(img, start, 3, [0, 0, 255], -1)
                cv2.line(img, start, far, [0, 255, 0], 1)
                cv2.line(img, end, far, [0, 255, 0], 1)

self._image_with_points = img

def show_processed_image(self):
    cv2.imshow('Image with points on fingers', self._image_with_points)
    cv2.waitKey(0)

def load_image_from_file(self, input_file_name=None):
    if input_file_name is None:
        input_file_name = self._input_file_name
    else:

```

```

        self.__input_file_name = input_file_name
        self.__image_without_points = cv2.imread(input_file_name)

    def save_image_to_file(self, output_file_name=None):
        if output_file_name is None:
            output_file_name = f'./training/test/{os.path.basename(self.__input_file_name)}'
        cv2.imwrite(output_file_name, self.__image_with_points)

```

```

#image_processing_app.py

```

```

import tkinter as tk
from tkinter import Button
from tkinter.filedialog import askopenfilename

class ImageProcessingApp(tk.Tk):
    def __init__(self, image_processor):
        self.__image_processor = image_processor
        tk.Tk.__init__(self)
        self.title("Hands photo processing application")
        self.geometry("500x260")
        self.resizable(False, False)

        self.__load_button = Button(self, text="Load image from file", command=self.load_from_file,
                                     width=61, height=2)
        self.__load_button.pack()

        self.__process_button = Button(self, text="Process image", command=self.process_image,
                                       state="disabled",
                                       width=61, height=2)
        self.__process_button.pack()

        self.__show_button = Button(self, text="Show processed image", command=self.show_processed_image,
                                     state="disabled", width=61, height=2)
        self.__show_button.pack()

        self.__save_button = Button(self, text="Save processed image to file", command=self.save_to_file,
                                     state="disabled", width=61, height=2)

        self.__save_button.pack()

        self.__exit_button = Button(self, text="Exit", command=self.exit, width=61, height=2)
        self.__exit_button.pack()

    def load_from_file(self):
        self.__image_processor.load_image_from_file(askopenfilename())
        self.__process_button["state"] = "normal"

    def process_image(self):
        self.__image_processor.find_points()
        self.__show_button["state"] = "normal"
        self.__save_button["state"] = "normal"

    def show_processed_image(self):
        self.__image_processor.show_processed_image()

    def save_to_file(self):
        self.__image_processor.save_image_to_file()

    def exit(self):
        self.quit()

```

```

#cmd_parser.py

import argparse

class ImageProcessorArgs:
    def __init__(self):
        parser = argparse.ArgumentParser(description="Program to find points - ends of fingers on hands photos")

        parser.add_argument("-i", "--input-file-name", type=str,
                            help="String that contain input image filename to load", dest="input_file_name")
        parser.add_argument("-o", "--output-file-name", type=str,
                            help="String that contain processed image filename to save", dest="output_file_name")
        parser.add_argument("-c", "--coefficient", type=float,
                            help="Approximation Poly DP Coefficient", dest="coefficient")
        parser.add_argument("-s", "--show", type=bool,
                            help="Is show final picture needed or now", dest="show_needed")

        args = parser.parse_args()

        self.__input_file_name = args.input_file_name
        self.__output_file_name = args.output_file_name
        self.__approx_poly_dp_coefficient = args.coefficient
        self.__show_needed = False if args.show_needed is None else args.show_needed

    @property
    def input_file_name(self):
        return self.__input_file_name

    @property
    def output_file_name(self):
        return self.__output_file_name

    @property
    def approx_poly_dp_coefficient(self):
        return self.__approx_poly_dp_coefficient

    @property
    def is_showing_picture_needed(self):
        return self.__show_needed

```