Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ МОРСКОЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет цифровых промышленных технологий
Кафедра вычислительной техники и информационных технологий

Отчет

По выполнению практического задания

по теме «**Применение методов лицевой биометрии в криминалистике**»

по дисциплине «**Прикладная информатика**».

Специальность: программное обеспечение вычислительной техники и информационных технологий

Выполнил:

Студент группы №20390 *Трапер Максим*

Дата выполнения отчета: 23.05.23

Дата сдачи отчета: 24.05.23

Подпись: _____

Проверил:

Профессор *Щеголева Н.Л.*

Подпись: _____

Санкт-Петербург

2023

Оглавление

Цель работы

Реализация алгоритма «автоматической генерации популяции скетчей» и проверка его эффективности в задаче сравнения фотографий со скетчами, составляемыми в криминалистике, и их модификациями.

Теоретические сведения

Индекс структурного подобия (**SSIM** – Structural Similarity Index Measure) – является одним из методов измерения схожести между двумя изображениями. SSIM-индекс это метод полного сопоставления, другими словами, он проводит измерение качества на основе исходного изображения. Оцениваются три фактора: яркостные изменения, изменениях контраста и потеря корелляции между изображениями.

Artist scetch – рисунок человеческого лица, составленный художником, со слов другого человека.

Кумулятивная сумма – это операция над последовательностью чисел, где новые числа формируются таким образом, что каждое последующее число основано на сумме всех предыдущих.

Ход работы

Ссылка на GitHub:

https://github.com/MaksimTraper/4cvtask

Код программы. Алгоритм:

Запуск программы осуществляется с генерации GUI начального окна. В нём есть поле ввода диапазона генерации параметров $P_i(i = \overline{1,3})$ и кнопка запуска алгоритма «автоматической генерации популяции скетчей».

```python
from AutoGeneratingScetches import AutoGenScetches
import tkinter as tk
new *
def startProgram():
    pDiapazon = int(pDiap.get())
    Generator = AutoGenScetches(amountScetches = 10, intervalRandom = pDiapazon)
    Generator.LoadPhotoAndScetch()
    #Generator.makeViewedScetch()
    Generator.GenerateP()
    Generator.MakeFirstScetchGeneration()
    Generator.MakeSecondScetchGeneration()
    Generator.CalculateSSIM()
    Generator.ShowResults()

startwindow = tk.Tk()
startwindow.geometry('250x150')
startwindow.title("Стартовое окно")

mes = tk.Label(startwindow, text='Введите диапазон изменения параметров P:')
btnStartPrg = tk.Button(startwindow, text="Запуск", command=startProgram)
pDiap = tk.Entry(startwindow)

mes.pack(ipadx=20, ipady=20)
pDiap.insert(0, '10')
pDiap.pack()
btnStartPrg.pack(ipadx=10, ipady=10, padx=10, pady=10)

startwindow.mainloop()
```

Далее, открывается проводник для выбора фотографии. По данным о выбранной фото загружается соответствующий ей скетч. Оба изображения переводятся в полутоновый.

```
Maksim
def loadPhotoPath(self):
    photoPath = fd.askopenfilename()
    root_proj = os.getcwd()
    rel_path = os.path.relpath(photoPath, root_proj)
    photoPath = rel_path
    rel_path = up(rel_path)
    photo = os.path.relpath(photoPath, rel_path)
    scetchPath = 'sketches/' + photo[0].upper() + '2' + photo[1:8:1] + '-sz1.jpg'
    return photoPath, scetchPath

Maksim
def LoadPhotoAndScetch(self):
    photoPath, scetchPath = self.loadPhotoPath()
    self.photo = cv2.imread(photoPath)
    self.grPhoto = self.cvt_to_gray(self.photo)
    self.scetch = cv2.imread(scetchPath)
    self.grScetch = self.cvt_to_gray(self.scetch)
    return

Maksim
def cvt_to_gray(self, photo):
    gr_photo = cv2.cvtColor(photo, cv2.COLOR_BGR2GRAY)
    return gr_photo
```

Затем, в границах задаваемого в стартовом окне диапазона, генерируются значения параметров P в количестве 10 для каждого. Т.е. в количестве скетчей, которые будут генерироваться в рамках первой популяции.

```
Maksim
def GenerateP(self):
    arr = [[], [], []]
    for i in range(0, 3):
        for y in range(0, self.amountScetches):
            num = 0
            while(num == 0):
                num = random.randint(-(self.intervalRandom), self.intervalRandom)
            arr[i] = np.append(arr[i], num)
    self.p1 = arr[0]
    self.p2 = arr[1]
    self.p3 = arr[2]
    return None
```

Далее следует этап генерации скетчей первой популяции на основе локальных изменений лица. Для исходного скетча в цикле (в каждом шаге берётся изначальный скетч, а получившийся в конце записывается в структуру данных со скетчами новой популяции) выполняется несколько шагов:

1) Проверяется P_1: если значение > 0, то из массива исходного скетча удаляются первые $P_1 - 1$ строк, иначе эти же строки дописываются «сверху» массива. Далее матрица масштабируется до исходных размеров, посредством линейной интерполяции (применяется метод resize библиотеки OpenCV). Первый случай привод к удлинению лица, второй – к обратному результату.

2) Проверяется P_2: если значение > 0, то из массива исходного скетча удаляются первые $P_1 - 1$ столбцов, иначе удаляются последние $P_1 - 1$ столбцов. Далее матрица масштабируется до исходных размеров. Первый случай привод к увеличению ширины лица, второй – к обратному результату.

3) Проверяется P_3: если значение > 0, то массив циклически сдвигается влево на $P_3 - 1$ столбцов, иначе вправо. Это привод к нарушению симметрии лица, относительно его центральной линии.

```
Maksim
87  def MakeFirstScetchGeneration(self):
88      l = len(self.p1)
89      for i in range (0, l):
90          if(self.p1[i] > 0):
91              a = np.delete(self.grScetch, np.s_[0:int(self.p1[i])-1], 0)
92              self.generation1.append(a)
93          else:
94              endSlice = int(abs(self.p1[i]))-1
95              sl = self.grScetch[0:endSlice]
96              a = np.insert(self.grScetch, 0, sl, 0)
97              self.generation1.append(a)
98          self.generation1[i] = cv2.resize(self.generation1[i], (200, 250), cv2.COLOR_BGR2GRAY, cv2.INTER_LINEAR)
99          fir = self.generation1[i]
100
101         if (self.p2[i] > 0):
102             border = int(self.p2[i])-1
103             self.generation1[i] = np.delete(self.generation1[i], np.s_[0:border], 1)
104         else:
105             border = int(abs(self.p2[i]))+1
106             r_border = 200 - border
107             self.generation1[i] = np.delete(self.generation1[i], np.s_[r_border:200], 1)
108         self.generation1[i] = cv2.resize(self.generation1[i], (200, 250), cv2.COLOR_BGR2GRAY)
109         sec = self.generation1[i]
110
111         if (self.p3[i] > 0):
112             numForRoll = int(abs(self.p3[i])-1)
113             arr1 = self.generation1[i][0:250, numForRoll:]
114             arr2 = self.generation1[i][0:250, :numForRoll]
115             self.generation1[i] = np.concatenate((arr1, arr2), axis=1)
116         else:
117             self.generation1[i] = np.roll(self.generation1[i], int(abs(self.p3[i])-1))
118     return None
```

Далее, на основе K скетчей первой популяции, генерируются K новых скетчей второй популяции. Генерация осуществляется посредством куммулятивной суммы входящих скетчей по формуле:

$$\tilde{S}^{(k)} = \frac{\sum_{j=1}^{k} S^{(j)}}{k}, \text{ для } k = 1,2,..,K.$$

```
Maksim
120     def MakeSecondScetchGeneration(self):
121         l = len(self.p1)
122         for i in range(0, l):
123             self.generation2.append(self.grScetch)
124         self.generation2 = [n.astype('int') for n in self.generation2]
125         for i in range(0, l):
126             for y in range(0, i+1):
127                 self.generation2[i] += self.generation1[y]
128             self.generation2[i] = self.generation2[i]/(i+2)
129         self.generation2 = [n.round() for n in self.generation2]
130         return None
```

Процесс генерации обеих популяций также продублирован на рисунке ниже.

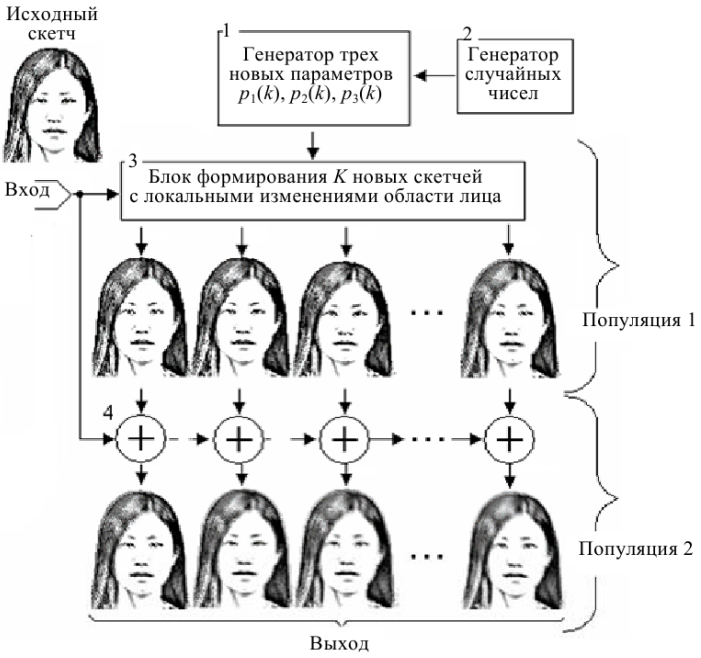


Рис: процесс генерации двух новых популяций скетчей

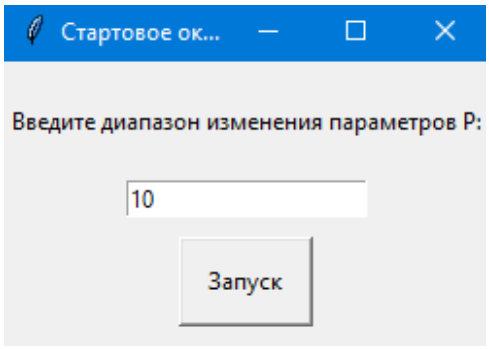
В качестве последнего этапа, рассчитывается SSIM для исходной фотографии в полутоне и: 1) исходного скетча в полутоне; 2) скетчей первой популяции; 3) скетчей второй популяции

```
Maksim
132     def CalculateSSIM(self):
133         l = len(self.p1)
134         self.generation2 = [n.astype('uint8') for n in self.generation2]
135         global ssimor
136         ssimor = ssim(self.grScetch, self.grPhoto)
137         for i in range(0, l):
138             self.ssim_gen1.append(ssim(self.generation1[i], self.grPhoto))
139             self.ssim_gen2.append(ssim(self.generation2[i], self.grPhoto))
140         return None
```

Результаты работы:

Для демонстрации работы программы была выбрана база данных CUHK (CUFS).

Стартовое окно. Здесь доступен ввод диапазона генерации значений параметров $P_i(i = \overline{1,3})$



Далее, для демонстрации итогового результата, появляется три окна. В первом окне демонстрируются исходные фото и скетч, а также график сравнения вычисленного SSIM между исходной фотографией и: 1) исходным скетчем (красная линия); 2) скетчами первой популяции (зелёная линия); 3) скетчами второй популяции (синяя линия)

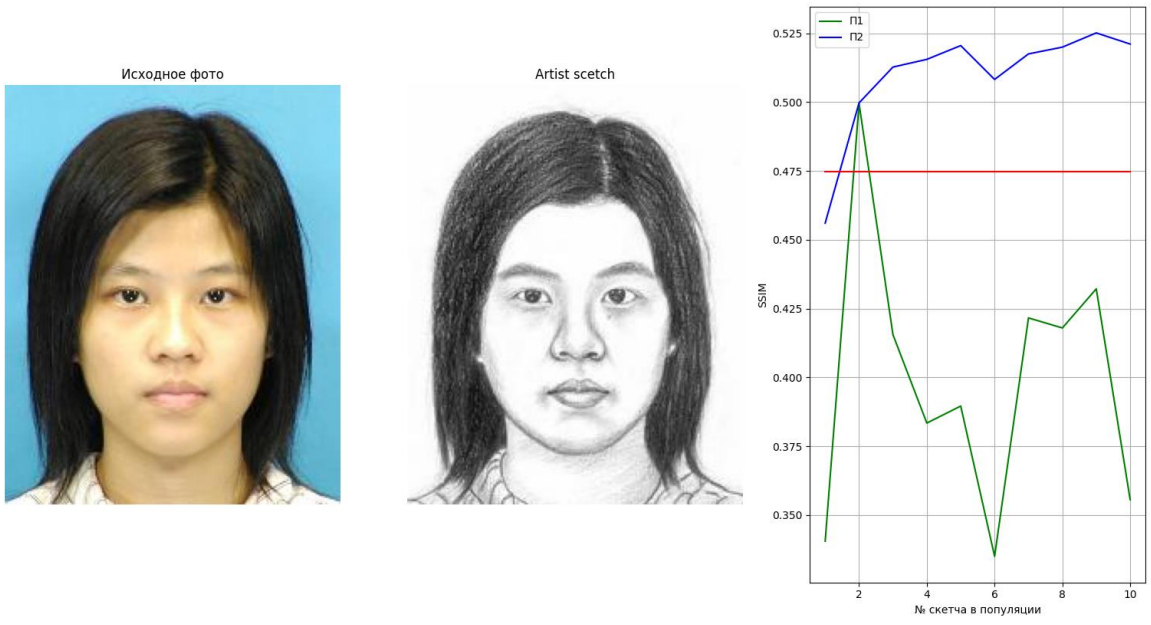


Рис: окно №1

Во втором и третьем окнах выводятся, соответственно, все скетчи первой и второй популяций для наглядной демонстрации, относительно каких скетчей вычислялся SSIM.

Во втором окне над каждым скетчем написаны значения параметров P, использовавшихся при локальных изменениях.

Рис: окно №2

Рис: окно №3

Вывод:

Посредством реализации алгоритма, была показана его эффективность. Сравнение скетчей второй популяции, основанных на различных спецификациях скетча (т.е., с локальными изменениями лица), с исходным скетчем с помощью SSIM показало, что такой способ генерации приводит к росту степени соответствия, между парами оригинальное фото – скетчи второй модификации, с каждым новым скетчем второй популяции.

Это позволяет использовать метод в криминалистике для повышения эффективности поиска людей по фотографиям, на основе разных фотороботов.