Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчет по лабораторной работе №1

Дисциплина: Методы и средства цифровой обработки информации

Выполнил
студент гр. 13541/2 _____ Зобков Д. А.
(подпись)

Преподаватель _____ Абрамов Н. А.
(подпись)

“___”_____ 2017 г.

Санкт-Петербург
2017 г.

СОДЕРЖАНИЕ

1 Техническое задание

- Подобрать некачественное (засвеченное) изображение в цветовом формате RGB;
- Преобразовать его при помощи математического алгоритма в черно-белый формат;
- Получить цветовую гистограмму изображения;
- При помощи методов улучшения гистограмм получить более качественные изображения в черно-белом формате.

2 Ход работы

Для выполнения работы был разработан класс ShadeFix с использованием библиотек OpenCV и Matplotlib на языке Python (листинг А.1 на с. 10). Пример его использования приведен в листинге А.2, а использованные зависимости для virtualenv — в листинге А.3.

2.1 Преобразование цветного изображения в черно-белое

Исходное изображение представлено на рис. 2.1.

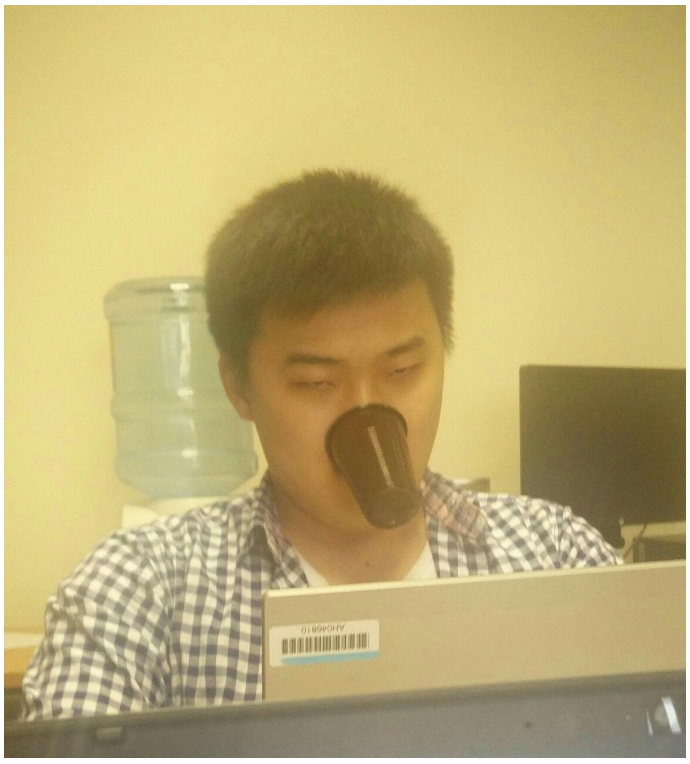

Рис. 2.1. Некачественно цветное изображение в формате RGB

Для преобразования цветного изображения в черно-белое воспользуемся формулой (2.1).

$$Gray_{x,y} = 0.299 \times R_{x,y} + 0.587 \times G_{x,y} + 0.114 \times B_{x,y} \quad (2.1)$$

В классе ShadeFix преобразование выполняется с помощью метода `toGreyscale`.

На рис. 2.2 показано изображение, полученное в результате работы программы.

Рис. 2.2. Черно-белое изображение, полученное в результате работы программы

Для построения гистограммы изображения используется метод `makeHistogram`.

Гистограмма полученного изображения представлена на рис. 2.3 на следующей странице.

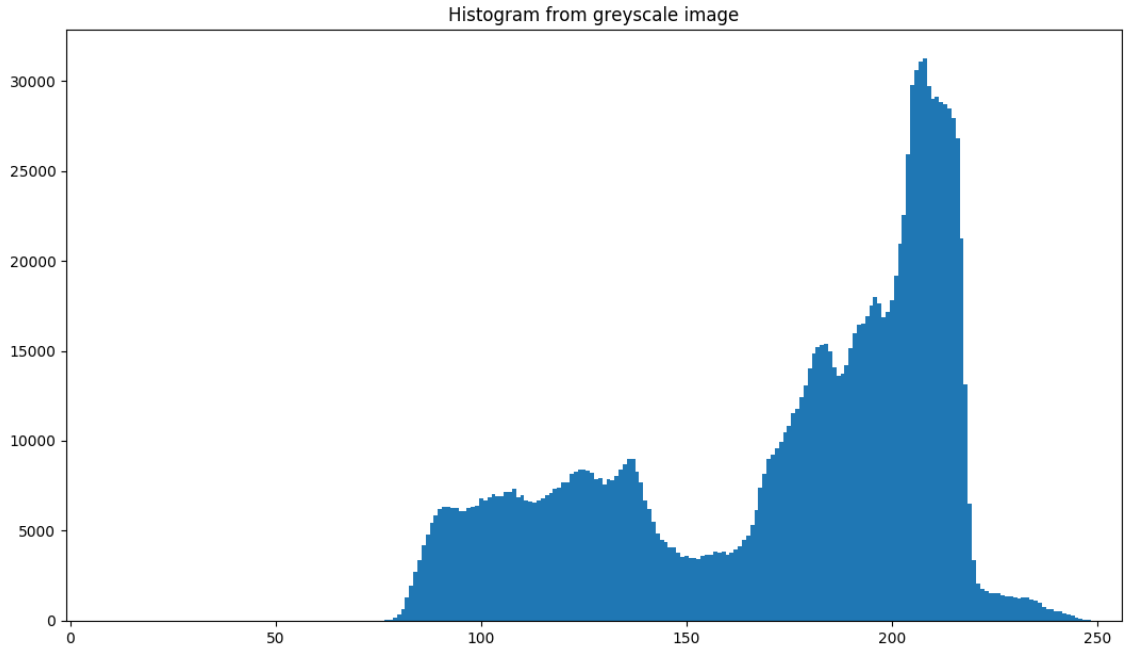

Рис. 2.3. Исходная гистограмма черно-белого изображения

2.2 Линейное растяжение гистограммы изображения

При выполнении линейного растяжения пренебрежем малыми цветами (порог 5%). Для этого используется метод `normalizeShadeMap`. Результирующая гистограмма представлена на рис. 2.4.

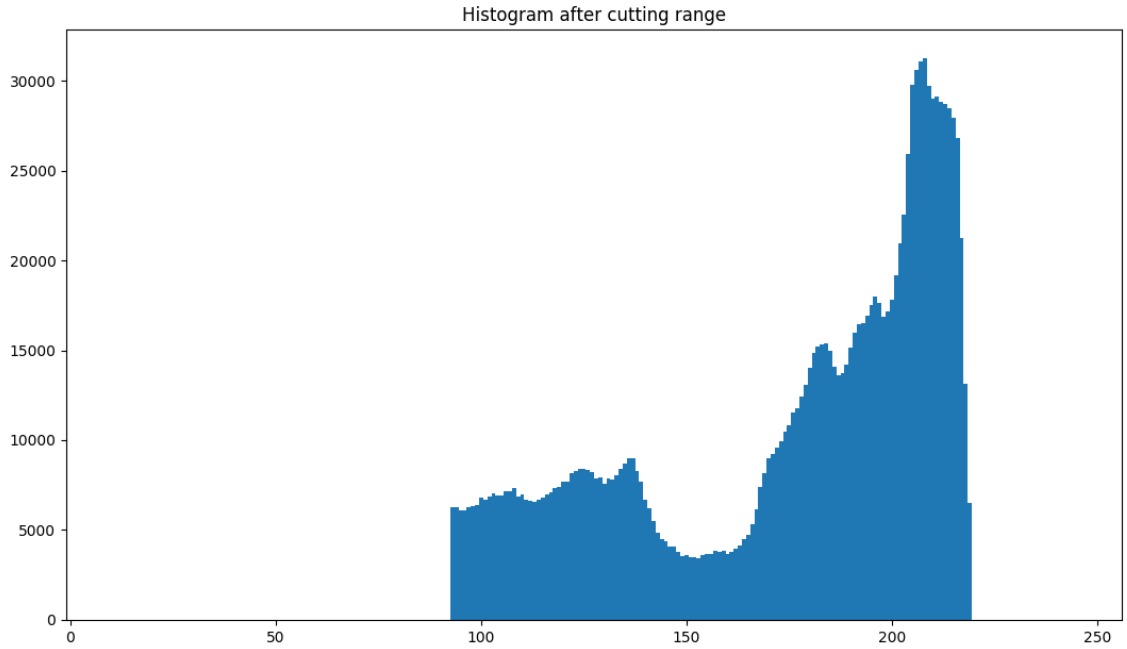

Рис. 2.4. Результирующая гистограмма черно-белого изображения

Улучшим качество изображения с помощью метода линейного растяжения. Для этого воспользуемся формулой (2.2), где $f_{x,y}^{out}$ — значение оттенка нового пикселя, $f_{x,y}^{in}$ — значение оттенка старого пикселя, a и b — нижний и верхний границы гистограммы, c и d — новые границы гистограммы. В классе ShadeFix данная формула реализована в методе `linearStretching`.

$$f_{x,y}^{out} = (f_{x,y}^{in} - a) * \frac{d-c}{b-a} + c \tag{2.2}$$

Полученная в результате данного преобразования гистограмма приведена на рис. 2.5, а результирующее изображение — на рис. 2.6 на следующей странице.

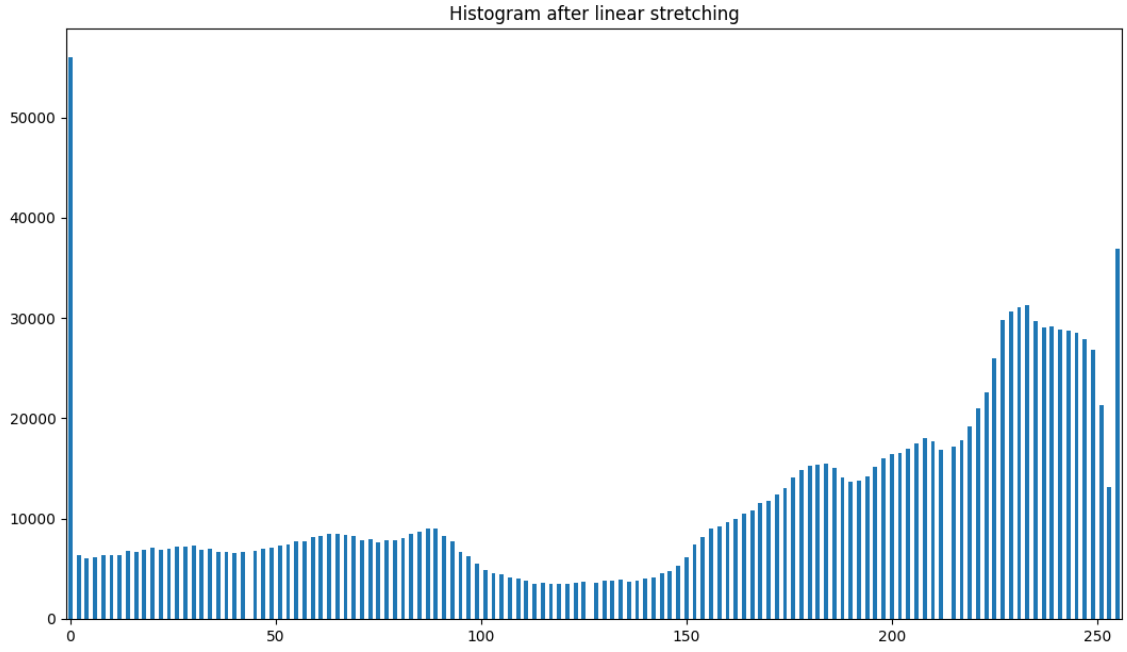

Рис. 2.5. Гистограмма с применением метода линейного растяжения

Рис. 2.6. Изображение, полученное в ходе растяжения гистограммы (справа), в сравнении с исходным

2.3 Эквализация гистограммы изображения

Попробуем улучшить исходное изображение путем эквализации его гистограммы. Для этого воспользуемся формулой (2.3), где S_k — элемент LookUp таблицы (хранящий новые значения оттенков для замены старых), n_j — число пикселей с оттенком j, n — число всех пикселей. В классе ShadeFix данная формула реализована в методе `histogramEqualization`.

$$S_k = \sum_{j=0}^{k} \frac{n_j}{n} \tag{2.3}$$

Полученная в результате данного преобразования гистограмма приведена на рис. 2.7, а результирующее изображение — на рис. 2.8 на следующей странице.

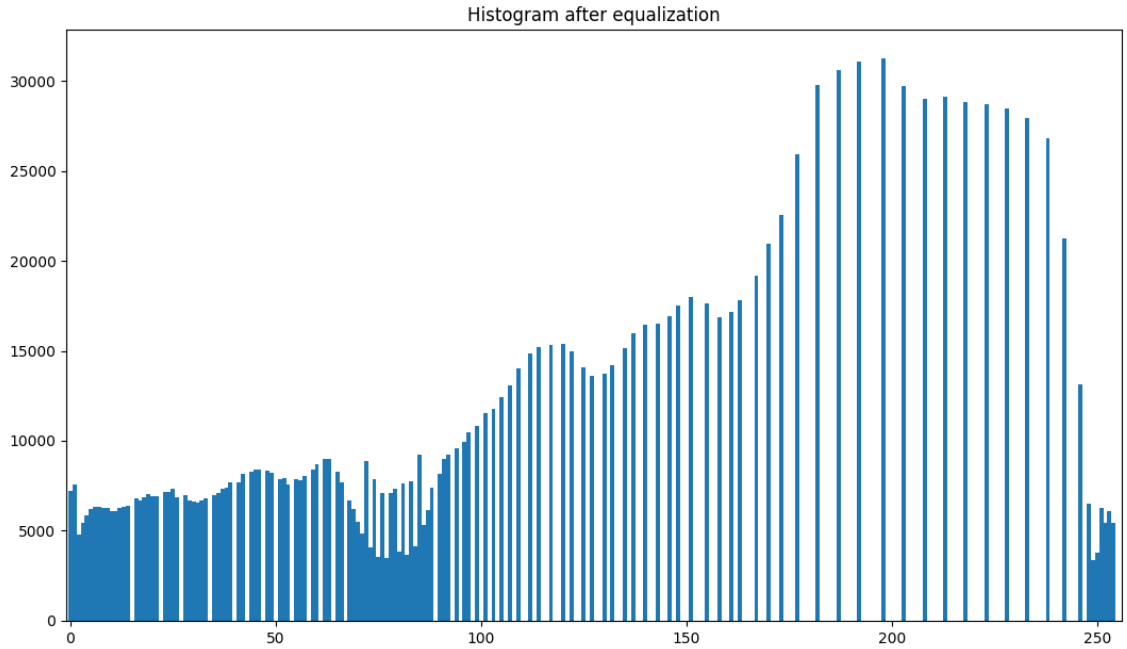

Рис. 2.7. Гистограмма с применением метода эквализации гистограммы

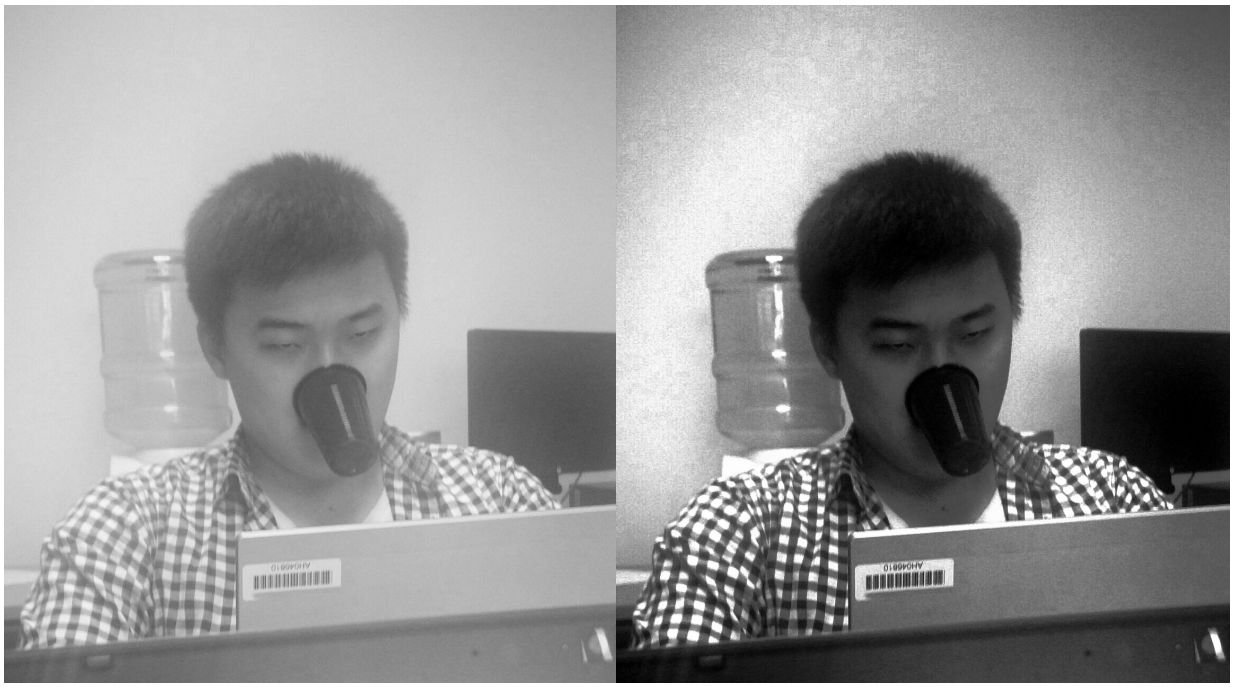

Рис. 2.8. Изображение, полученное в ходе эквализации гистограммы (справа), в сравнении с исходным

3 Выводы

В ходе данной работы был разработан класс для улучшения засвеченного черно-белого изображения с помощью методов линейного растяжения и эквализации гистограммы.

При исследовании гистограмм после линейного растяжения и эквализации гистограммы можно заметить, что метод линейного растяжения „растягивает“ исходную гистограмму на весь диапазон оттенков, а метод эквализации гистограммы старается обеспечить средний уровень количества пикселей по всей гистограмме (так, в случае, когда на один оттенок приходится большое количество пикселей, расстояние до соседних ненулевых оттенков сравнимо больше, чем в случае оттенка с малым числом пикселей).

В результате выполнения работы было обнаружено, что линейное растяжение неплохо уменьшает засвеченность изображения, однако для этого необходимо, чтобы в гистограмме отсутствовали оттенки на ее границах (в данном случае 0 и 255), поскольку в таком случае действенность этого метода резко ухудшается. Эквализация гистограммы же может привести к сильному затемнению или засветлению изображения (создать „шумы“). Обычно подобный эффект проявляется на почти однотонных участках исходного изображения (из близких оттенков). Вызвана данная ситуация перераспределением пикселей, обеспечивающих плавный переход, к разным контрастным оттенкам (наблюдаемые на результирующей гистограмме расстояния между соседними ненулевыми оттенками), что приводит к увеличению контрастности и ухудшению плавности.

Приложение А Исходный код программы

```
#!/usr/bin/env python

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

class ShadeFix:
    """
    Class for performing linear stretching and histogram equalization
   on greyscale image

    :ivar    inImg:    Loaded image, converted to greyscale
    :vartype inImg:    numpy
    :ivar    height:   Height of image
    :vartype height:   int
    :ivar    width:    Width of image
    :vartype width:    int
    :ivar    origImg:  Backup of original greyscale image
    :vartype origImg:  numpy
    :ivar    shadeMap: Map with the amount of each shade
    :vartype shadeMap: numpy
    """
    def __init__(self, inImg):
        # Load image
        self.inImg = cv.imread(inImg)
        # Rows
        self.height = self.inImg.shape[0]
        # Columns
        self.width = self.inImg.shape[1]
        # RGB to Greyscale image
        if len(self.inImg.shape) == 3: self.toGreyscale()
        # Greyscale image backup
        self.origImg = self.inImg
        # Shade Map
        self.shadeMap = self.getShadeMap()

    def toGreyscale(self):
        """
        Converting RGB image to greyscale
        """
        # Get color arrays
        red   = self.inImg[...,2]
        green = self.inImg[...,1]
        blue  = self.inImg[...,0]

        # Fill array with shades of grey
        outImg = np.zeros((self.height, self.width))
        outImg[...] = 0.299 * red + 0.587 * green + 0.114 * blue

        # Round result shades
        outImg = np.round(outImg)
```

```
52          # Update image
            self.inImg = outImg
54
        def linearStretching(self):
56          """
            Performing linear stretching on greyscale image
58          """
            # Get bounds of histogram
60          a = np.nonzero(self.shadeMap)[0][0]
            b = np.nonzero(self.shadeMap)[0][-1]
62
            # Linear stretching
64          resImg = np.zeros((self.height, self.width))
            resImg[...] = (self.inImg[...] - a) * (255 / (b - a))
66
            # Fix out of range values
68          resImg[resImg < 0] = 0
            resImg[resImg > 255] = 255
70          resImg = np.round(resImg)

72          # Update image and shade map
            self.inImg = resImg
74          self.shadeMap = self.getShadeMap()

76      def histogramEqualization(self):
            """
78          Performing histogram equalization on greyscale image
            """
80          # Create shades replace map and fill it
            shades = np.zeros(256)
82          pixelCount = self.height * self.width
            for k in range(0, shades.size - 1):
84              shades[k] = np.sum(self.shadeMap[0:k]) / pixelCount

86          # Histogram stretching for the whole range
            shades[...] *= 255
88
            # Replace pixels according to shade map
90          resImg = np.zeros((self.height, self.width))
            resImg[...] = shades[self.inImg[...].astype(int)]
92
            # Update image and shade map
94          self.inImg = resImg
            self.shadeMap = self.getShadeMap()
96
        def makeHistogram(self, title, path):
98          """
            Creating histogram of image and save to file
100
            :param title: Title of histogram
102         :type  title: str
            :param path:  Save path
104         :type  path:  str
```

```
        """
106     # Create plot
        fig = plt.figure(figsize=(12.80, 7.20), dpi=100)
108     plt.bar(np.arange(256), self.shadeMap, 1)
        plt.xlim(-1, 256)
110     plt.title(title)

112     # Save plot
        fig.savefig(path)
114
    def normalizeShadeMap(self):
116     """
        Deleting about 5% of histogram pixels from corners
118     """
        check, a, b = 0, 0, 255
120     while check < 0.05 * self.width * self.height:
            # Cut from left side
122         if self.shadeMap[a] <= self.shadeMap[b]:
                check += self.shadeMap[a]
124             self.shadeMap[a] = 0
                a += 1
126         # Cut from right side
            else:
128             check += self.shadeMap[b]
                self.shadeMap[b] = 0
130             b -=1

132 def getShadeMap(self):
        """
134     Creating map with the amount of each shade

136     :return: Shade map
        :rtype:  numpy
138     """
        return np.bincount(self.inImg.astype(int).flat, minlength=256)
140
    def saveImage(self, path):
142     """
        Saving image to file
144
        :param path: Save path
146     :type  path: str
        """
148     cv.imwrite(path, self.inImg)

150 def restoreImage(self):
        """
152     Restoring original greyscale image and shade map from backup
        """
154     self.inImg = self.origImg
        self.shadeMap = self.getShadeMap()
```

Листинг А.1. shadefix.py

```
#!/usr/bin/env python

from shadefix import ShadeFix
import os, sys

# Create output directory
dirname = "out"
if not os.path.exists(dirname): os.mkdir(dirname)

# Load image
if len(sys.argv) > 1:
        img = ShadeFix(sys.argv[1])
else:
        img = ShadeFix("test.jpg")

imgName  = [dirname + "/grey{}.png".format(i + 1) for i in range(3)]
histName = [dirname + "/hist{}.png".format(i + 1) for i in range(4)]

# Save greyscale image
img.saveImage(imgName[0])

# Save greyscale image histogram
img.makeHistogram("Histogram from greyscale image", histName[0])

# Save normalized histogram
img.normalizeShadeMap()
img.makeHistogram("Histogram after cutting range", histName[1])

# Save image and histogram after linear stretching
img.linearStretching()
img.saveImage(imgName[1])
img.makeHistogram("Histogram after linear stretching", histName[2])

# Restore original greyscale image
img.restoreImage()

# Save image and histogram after histogram equalization
img.histogramEqualization()
img.saveImage(imgName[2])
img.makeHistogram("Histogram after equalization", histName[3])
```

Листинг А.2. main.py

```
matplotlib==2.0.2
numpy==1.12.1
opencv-python==3.3.0.10
```

Листинг А.3. Файл с использованными зависимостями для virtualenv