Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №3**

**Дисциплина: Алгоритмы цифровой обработки мультимедиа**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_К.В.Стасюк

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А. Крамаренко

Лабораторная работа №3

Методы фильтрации и размытия изображения.

Размытие изображения — это процесс обработки изображения, при котором каждый пиксель на картинке изменяется так, чтобы он был более похожим на своих соседей и не сильно выделялся по яркости или цвету. Если пиксель сильно отличается от своих соседей, то это может быть интерпретировано как шум на изображении. Размытие помогает сгладить такие шумы и сделать изображение более однородным.

Чтобы начать процесс размытия, обычно изображение сначала преобразуется в черно-белый формат. В этом формате каждый пиксель представлен только одним значением яркости, которое является неотрицательным целым числом.

Алгоритм кратко будет выглядеть следующим образом:

1. установить размер матрицы свертки и среднее квадратичное отклонение;
2. заполнить матрицу свертки значениями функции Гаусса с мат.

ожиданием, равным координатам центра матрицы;

1. нормировать матрицу так, чтобы сумма элементов равнялась 1;
2. создать копию изображения;
3. для каждого внутреннего пикселя копии рассчитать новое значение
4. насыщенности пикселя по формуле (1) и ЗАПИСАТЬ это значение в пиксель нового изображения

**Задание 1**. Выполнить пункты 1 и 2 алгоритма, то есть построить

матрицу Гаусса. Просмотреть итоговую матрицу для размерностей 3, 5, 7.

Для начала мы задаем значение размера ядра свертки. Обычно он задается как нечетное число (например, 3, 5, 7), чтобы матрица имела центр. Чем больше размер ядра, тем более широкий радиус учета пикселей, и, следовательно, более сильное размытие.Математически, больший размер ядра означает большее количество пикселей, участвующих в вычислениях, и большее влияние каждого пикселя на окончательное значение размытого пикселя.

Далее матрица может быть инициализирована любыми значениями, в нашем случае-единицами.

После чего мы заполняем ее значениями, которые вычисляются по формуле Гаусса.

Здесь 𝑎, 𝑏 – координаты центрального элемента матрицы(математическое ожидание двумерной случайной величины).

В качестве 𝑥, 𝑦 выбираем все индексы всех элементов матрицы поочередно.

Значение 𝜎 будем считать параметром размытия по Гауссу. Этот параметр определяет "ширину" распределения Гаусса и, следовательно, степень размытия. Чем больше значение стандартного отклонения, тем менее интенсивное размытие. Математически, большее стандартное отклонение приводит к более плавному уменьшению весов пикселей с увеличением расстояния от центрального пикселя, что создает более "мягкое" размытие.

Изображение выглядит как текст, снимок экрана, меню, Шрифт

Автоматически созданное описание

Рисунок 1 – полученные значения матриц.

**Задание 2.** Нормировать полученную матрицу Гаусса. Протестировать

результаты на матрицах из предыдущего пункта.

Нормирование матрицы состоит из двух шагов:

1)Подсчет суммы всех элементов матрицы

2) Деление всех элементов на полученную сумму

Изображение выглядит как текст, снимок экрана, меню, Шрифт

Автоматически созданное описание

Рисунок 2 – значения нормализованных матрицы.

**Задание 3**. Реализовать фильтр Гаусса средствами языка python.

После нормализации матрицы ядра мы можем приступить к заключительному шагу-операции свертки.

Свертка - это математическая операция, которая применяет ядро к изображению для получения нового изображения, в котором каждый пиксель вычисляется на основе весов в ядре и значений пикселей в окрестности.

Можем записать следующую формулу:

где

B- входное изображение.

ker - это матрица (ядро) свертки

Основные шаги операции свертки:

1)Ядро свертки перемещается по изображению.

2)Для каждого пикселя в изображении ядро позиционируется так, чтобы его центр совпадал с текущим пикселем.

3)Выполняется элементное умножение (поэлементное перемножение) значений пикселей изображения и значений соответствующих элементов ядра свертки.

4)Результаты элементных умножений суммируются, и эта сумма становится новым значением центрального пикселя в результирующем изображении.

Процесс повторяется для каждого пикселя в исходном изображении, и на выходе получается новое изображение.

Изображение выглядит как природа, снег, черно-белый, пейзаж

Автоматически созданное описание

Рисунок 3 – реализация фильтра Гаусса.

**Задание 4.** Применить данный фильтр для двух разных значений

среднего квадратичного отклонения и двух разных размерностей матрицы свертки, сравнить результаты для ОДНОГО изображения.

На рисунках ниже можно увидеть, что при большем значении ядра матрицы свертки эффект размытия увеличивается, при меньшем соответственно уменьшается. А вот значение стандартного отклонения работает противоположно, при увеличении отклонения эффект размытия будет уменьшаться, при уменьшении-увеличиваться.

Изображение выглядит как снег, черно-белый, снимок экрана, пейзаж

Автоматически созданное описание

Рисунок 4 - реализация для разных значений параметров.

**Задание 5.** Реализовать размытие Гаусса встроенным методом библиотеки OpenCV, сравнить результаты с Вашей реализацией.

Для данного задания была использована функция cv2.GaussianBlur

Вот основные аргументы функции:

src: Исходное изображение, которое нужно размыть.

ksize: Размер ядра (окна), которое будет использоваться для вычисления размытия.

sigmaX: Стандартное отклонение по горизонтали.

dst (необязательный): Выходное изображение, в которое будет записан результат. Если не указано, результат будет записан в src.

sigmaY (необязательный): Стандартное отклонение по вертикали. Если не указано, считается равным sigmaX.

borderType (необязательный): Тип границы, который определяет, как обрабатывать пиксели на границах изображения. По умолчанию используется cv2.BORDER\_DEFAULT.

Можно заметить, что результаты получились идентичны.

Изображение выглядит как снег, снимок экрана, гора, природа

Автоматически созданное описание

Рисунок 5 - реализация размытия с помощью библиотеки Opencv.

**Задание 6 (самостоятельно)** Реализовать размытие Гаусса средствами любого другого языка программирования.

Результат реализации на с++. Здесь в консоли представлена исходная матрица, нормализованная и полученная после размытия. Алгоритм написан по описанию выше.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 5 - реализация размытия на ЯП с++.

**Листинг программы**

import cv2

import numpy as np

def main():

#задание 4- применение размытия Гаусса

img = cv2.imread("test4.jpg", cv2.IMREAD\_GRAYSCALE)

#стандартное отклонение

standard\_deviation = 100

standard\_deviation1 = 200

standard\_deviation2 = 50

#размер ядра(матрицы)

kernel\_size = 5

kernel\_size1 = 3

kernel\_size2 = 7

#размытие Гаусса (усредняет значения пикселей в соответствии с их расположением относительно центрального пикселя и весами)

imgBlur\_1 = AnotherGaussianBlur(img, kernel\_size, standard\_deviation)

imgBlur\_2 = AnotherGaussianBlur(img, kernel\_size1, standard\_deviation1)

imgBlur\_3 = AnotherGaussianBlur(img, kernel\_size2, standard\_deviation2)

cv2.imshow('Original\_image', img)

cv2.imshow(str(kernel\_size) + 'x' + str(kernel\_size) + ' and deviation ' + str(standard\_deviation), imgBlur\_1)

cv2.imshow(str(kernel\_size1) + 'x' + str(kernel\_size1) + ' and deviation ' + str(standard\_deviation1), imgBlur\_2)

cv2.imshow(str(kernel\_size2) + 'x' + str(kernel\_size2) + ' and deviation ' + str(standard\_deviation2), imgBlur\_3)

# Задание 5 - Реализация размытие Гаусса встроенным методом OpenCV

imgBlur\_CV2 = cv2.GaussianBlur(

img, (kernel\_size, kernel\_size), standard\_deviation)

cv2.imshow('Blur\_by\_CV2', imgBlur\_CV2)

cv2.waitKey(0)

def AnotherGaussianBlur(img, kernel\_size, standard\_deviation):

#задание 1- построение матрицы гаусса

#построение начальной матрицы ядра свертки из единиц.

kernel = np.ones((kernel\_size, kernel\_size))

a = b = (kernel\_size+1) // 2 #координаты центрального элемента матрицы

# построение матрицы свёртки

for i in range(kernel\_size):

for j in range(kernel\_size):

kernel[i, j] = gauss(i, j, standard\_deviation, a, b) # вычисление функции Гаусса для каждого элемента матрицы

print("Матрица ядра свертки для ядра=",kernel\_size)

print(kernel)

# Задание 2 - Нормализация полученной матрицы

sum = 0

for i in range(kernel\_size):

for j in range(kernel\_size):

sum += kernel[i, j]

for i in range(kernel\_size):

for j in range(kernel\_size):

kernel[i, j] /= sum

print("Нормализованная матрица ядра свертки:")

print(kernel)

# применение операции свёртки

imgBlur = Convolution(img, kernel)

return imgBlur

# реализация операции свёртки изображения

def Convolution(img, kernel):

kernel\_size = len(kernel)

imgBlur = img.copy()

# Вычисляем начальные координаты для итераций по пикселям

x\_start = kernel\_size // 2

y\_start = kernel\_size // 2

# Начинаем проход по каждому пикселю изображения, исключая края, чтобы не "выходить за границы" изображения

for i in range(x\_start, imgBlur.shape[0] - x\_start):

for j in range(y\_start, imgBlur.shape[1] - y\_start):

val = 0

# Проходимся по каждому элементу ядра свертки(центр+-kernel\_size // 2)

for k in range(-(kernel\_size // 2), kernel\_size // 2 + 1):

for l in range(-(kernel\_size // 2), kernel\_size // 2 + 1):

# Умножаем значение текущего пикселя(центральный+смещение) изображения на соответствующий элемент ядра свертки и суммируем

val += img[i + k, j + l] \* kernel[k + (kernel\_size // 2), l + (kernel\_size // 2)]

# Значение val становится новым значением пикселя в результирующем изображении

imgBlur[i, j] = val

# Возвращаем результирующее изображение после свертки

return imgBlur

# реализация функции Гаусса

def gauss(x, y, omega, a, b):

omegaIn2 = 2 \* omega \*\* 2

m1 = 1/(np.pi \* omegaIn2)

m2 = np.exp(-((x-a) \*\* 2 + (y-b) \*\* 2)/omegaIn2)

return m1\*m2

main()