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**

**Отчёт**

**по лабораторной работе №3**

**Дисциплина: АЛГОРИТМЫ ЦИФРОВОЙ ОБРАБОТКИ МУЛЬТИМЕДИА**

**Тема: «Методы фильтрации и размытия изображения»**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К. А. Корнилов

Направление подготовки 02.03.02 Фундаментальная информатика и

информационные технологии

Направленность (профиль) Математическое и программное обеспечение

компьютерных технологий

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

**Тема:** Методы фильтрации и размытия изображения.

**Цель:** Реализация метода размытия изображения с помощью шума Гаусса.

**Ход работы:**

1. Была построена матрица Гаусса. Просмотрена итоговую матрицу для размерностей 3, 5, 7.

Для этого сначала создается пустая квадратная матрица по размерам ядра, затем вычисляются математические ожидания величин, равные значению индексов центральной ячейки. После этого ячейки матрицы заполняются значениями функциями Гаусса.

Код:

def gaussMatrix(kernel\_size, standard\_deviation=0.2):

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

a = b = (kernel\_size + 1) // 2

# Строим матрицу свёртки

for i in range(kernel\_size):

for j in range(kernel\_size):

kernel[i, j] = gaussFunc(i, j, standard\_deviation, a, b)

print(kernel)

print("//////////")

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

omega2 = 2 \* omega \*\* 2

m1 = 1 / (np.pi \* omega2)

m2 = np.exp(-((x-a) \*\* 2 + (y-b) \*\* 2) / omega2)

return m1 \* m2

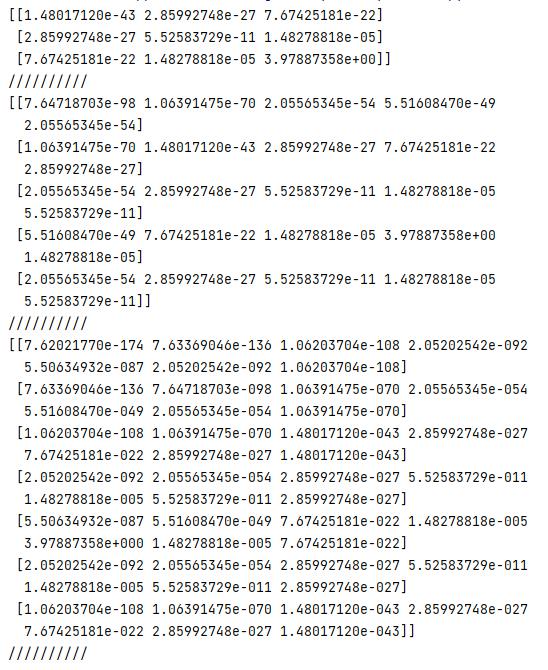


Рисунок 1 – Результаты создания матриц

1. Были нормированы полученные матрицы Гаусса.

Для этого вычисляется сначала сумма всех элементов матрицы, а затем каждый из элементов матрицы делится на полученную сумму. В итоге получается матрица с суммой элементов 1.

Код:

elementSum = np.sum(kernel)  
kernel = kernel/elementSum

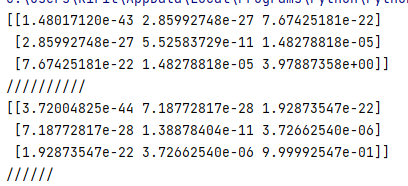


Рисунок 2 – Результат для матрицы 3x3

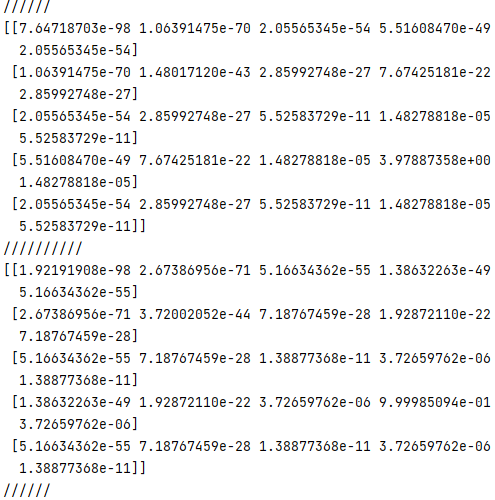


Рисунок 3 – Результат для матрицы 5x5

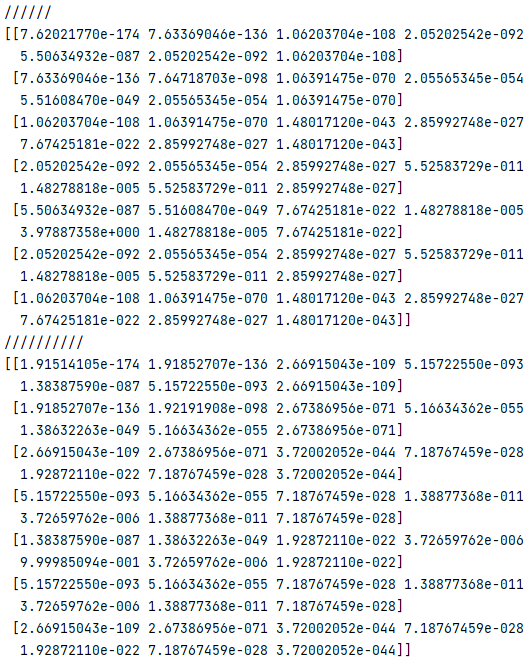


Рисунок 4 – Результат для матрицы 7x7

1. Был реализовать фильтр Гаусса средствами языка python.

Для этого сначала создается ядро для создания шума. Затем вычисляются стартовые координаты пикселей, как половина размера ядра. После этого запускается основной цикл, который применяет операцию свертки к каждому пикселю.

Код:

def gaussBlur(img,kernelSize=3,standardDeviation=0.2):  
 kernel = gaussMatrix(kernelSize,standardDeviation)  
 imgBlur = img.copy()  
  
 *#Заводим стартовые индексы* xStart = kernelSize // 2  
 yStart = kernelSize // 2  
 for i in range(xStart, imgBlur.shape[0] - xStart):  
 for j in range(yStart, imgBlur.shape[1] - yStart):  
 *# операция свёртки* val = 0  
 for k in range(-(kernelSize // 2), kernelSize // 2 + 1):  
 for l in range(-(kernelSize // 2), kernelSize // 2 + 1):  
 val += img[i + k, j + l] \* kernel[k + (kernelSize // 2), l + (kernelSize // 2)]  
 imgBlur[i, j] = val  
 return imgBlur

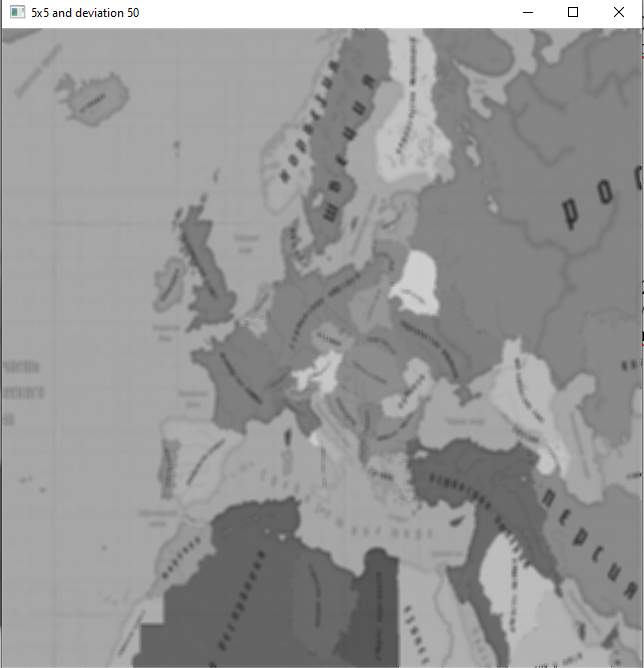


Рисунок 5 – Результат работы на ядре 5x5 и среднеквадратичном отклонении 50

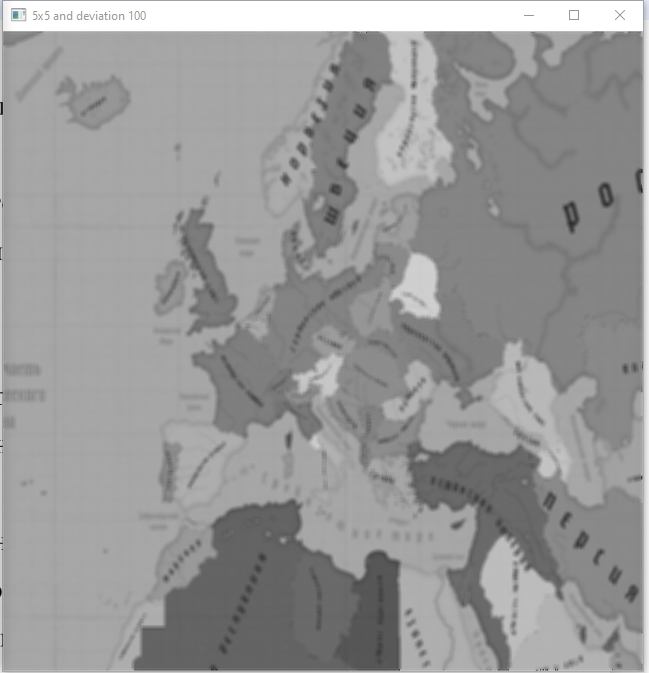


Рисунок 6 – Результат работы на ядре 5x5 и среднеквадратичном отклонении 100

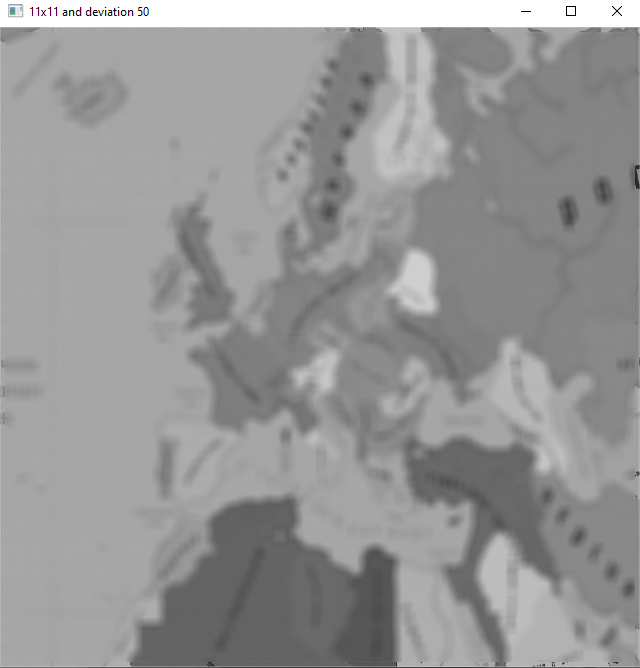


Рисунок 7 – Результат работы на ядре 11x11 и среднеквадратичном отклонении 50



Рисунок 8 – Результат работы на ядре 11x11 и среднеквадратичном отклонении 100

1. Реализовано размытие Гаусса встроенным методом библиотеки OpenCV, сравнены результаты с моей реализацией.

Для этого вызывается метод GaussianBlur с аргументами в виде изображения, размеров ядра и среднеквадратичного отклонения

Код:

def applyGaussianFilterOpencv(image, size, sigma):  
 return cv2.GaussianBlur(image, (size, size), sigma)

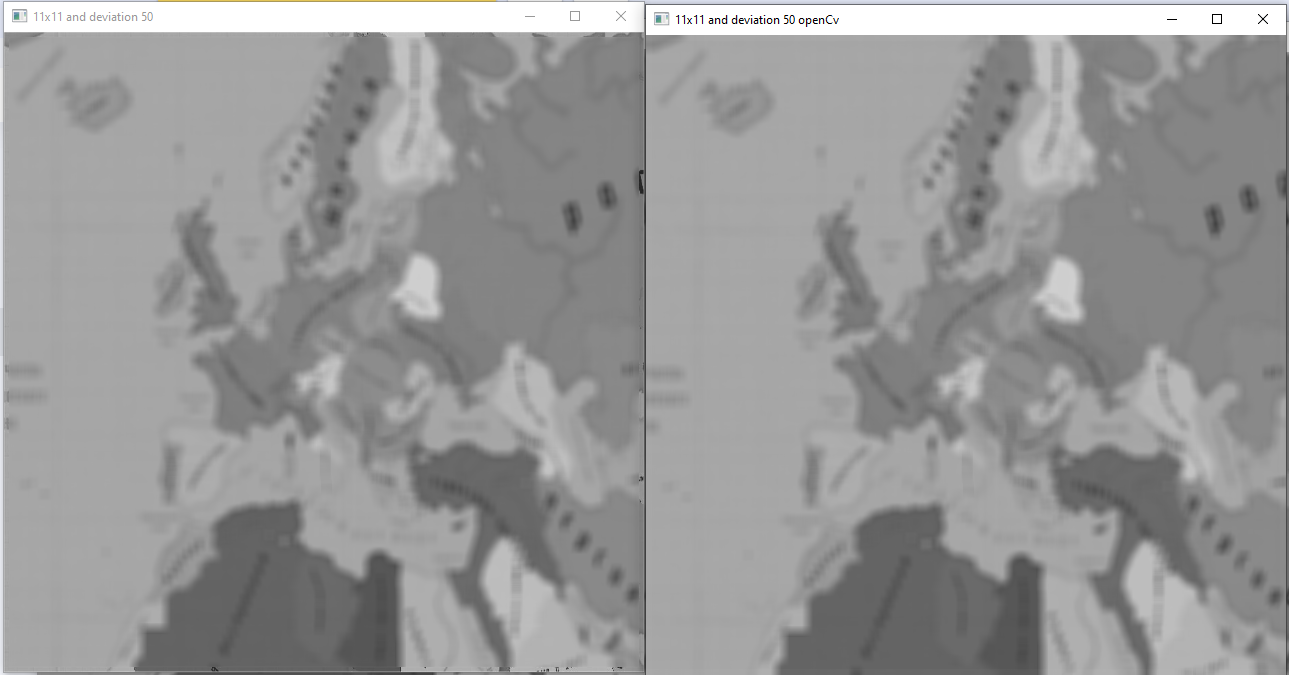


Рисунок 9 – Сравнение результатов работы

1. Реализовано размытие Гаусса средствами языка Kotlin. Для этого используется модуль scrimage, содержащий функции для работы с изображениями

Код:  
import com.sksamuel.scrimage.ImmutableImage  
import com.sksamuel.scrimage.filter.GrayscaleFilter  
import com.sksamuel.scrimage.nio.JpegWriter  
import com.sksamuel.scrimage.pixels.Pixel  
import java.io.File  
import java.lang.Math.pow  
class Gauss {  
 private fun gaussMatrix(kernelSize:Int=3, standardDeviation:Double=0.2): Array<Array<Double>> {  
 val kernel = Array(kernelSize) **{** Array(kernelSize) **{** 0.0 **} }** val a = (kernelSize + 1) / 2  
 val b = (kernelSize + 1) /2  
 //Строим матрицу свёртки  
 for(i in 0..<kernelSize) {  
 for(j in 0..<kernelSize) {  
 kernel[i][j] = gaussFunc(i, j, standardDeviation, a, b)  
 }  
 }  
 *print*(kernel)  
 *print*("//////////")  
 var elementSum = 0.0  
 kernel.*forEach* **{ it**.*forEach* **{** el**->**elementSum+=el **} }** for(i in 0..<kernelSize) {  
 for(j in 0..<kernelSize) {  
 kernel[i][j] = kernel[i][j]/elementSum  
 }  
 }  
 return kernel  
 }  
 private fun gaussFunc (x:Int, y:Int, omega:Double, a:Int, b:Int): Double {  
 val omega2 = 2 \* pow(omega,2.0)  
 val m1 = 1 / (Math.*PI* \* omega2)  
 val m2 = Math.exp(-(Math.pow((x - a).toDouble(),2.0) + Math.pow((y - b).toDouble(),2.0) / omega2))  
 return m1 \* m2  
 }  
 fun gaussBlur(img: ImmutableImage, kernelSize:Int=3, standardDeviation:Double=0.2): ImmutableImage? {  
 val kernel = gaussMatrix(kernelSize,standardDeviation)  
 val imgBlur = img.copy()  
 //Заводим стартовые индексы  
 val xStart = kernelSize / 2  
 val yStart = kernelSize / 2  
 for(i in xStart..<imgBlur.height - xStart) {  
 for (j in yStart..<imgBlur.width - yStart) {  
 var r = 0.0  
 var g = 0.0  
 var b = 0.0  
 // Применение ядра к соседним пикселям  
 for (kx in -kernelSize / 2..kernelSize / 2) {  
 for (ky in -kernelSize / 2..kernelSize / 2) {  
 val pixelX = (i + kx).*coerceIn*(0, imgBlur.width - 1)  
 val pixelY = (j + ky).*coerceIn*(0, imgBlur.height - 1)  
 val pixelColor = imgBlur.pixel(pixelX, pixelY)  
 val weight = kernel[kx + kernelSize / 2][ky + kernelSize / 2]  
 r += pixelColor.red() \* weight  
 g += pixelColor.green() \* weight  
 b += pixelColor.blue() \* weight  
 }  
 }  
 imgBlur.setPixel(  
 Pixel(  
 i,  
 j,  
 r.toInt().*coerceIn*(0, 255),  
 g.toInt().*coerceIn*(0, 255),  
 b.toInt().*coerceIn*(0, 255),  
 1  
 )  
 )  
 }  
 }  
 return imgBlur  
 }  
 fun BlurFuss(){  
 val imageFile = File("src/main/resources/test2.jpg")  
 val inputImage = ImmutableImage.loader().fromFile(imageFile)  
 val resizedImage = inputImage.scaleTo(640, 640)  
 val filtered = resizedImage.filter(GrayscaleFilter())  
 val writer = JpegWriter()  
 val kernel\_size = 11  
 val standard\_deviation = 100.0  
 val imgBlur1 = gaussBlur(filtered, kernel\_size, standard\_deviation)  
 val resultFile = imgBlur1?.output(writer, File("src/main/resources/test4.jpg"))  
 }  
}



Рисунок 9 – Пример результата работы

**Вывод:** Был реализован метод размытия изображения с помощью шума Гаусса средствами языков Python и Kotlin.

**1. Принцип операции размытия изображения и ее применение**

Размытие изображения — это операция, которая снижает резкость и детализацию изображения, сглаживая его пиксели. Этот процесс помогает уменьшить шум и улучшить визуальное восприятие, а также подготовить изображение для дальнейших операций обработки, таких как детекция краев или сегментация. Размытие может быть полезно в различных приложениях, включая:

* Устранение шумов в изображениях.
* Подготовка изображений для анализа.
* Улучшение качества при отображении изображений.

**2. Операция свертки матрицы**

Свертка — это математическая операция, которая комбинирует два массива (например, изображение и ядро) для создания нового массива. В контексте обработки изображений свертка применяется для изменения значений пикселей, основываясь на значениях окружающих пикселей. Операция свертки включает:

* Применение ядра (матрицы свертки) к каждому пикселю изображения.
* Вычисление новой интенсивности пикселя на основе значений окружающих пикселей и весов из ядра.

**3. Матрица свертки в размытии по Гауссу**

Матрица свертки для размытия по Гауссу (ядро Гаусса) строится на основе функции Гаусса. Она имеет следующие характеристики:

* Центральный пиксель имеет максимальное значение, и значения уменьшаются по мере удаления от центра.
* Значения в матрице нормализованы так, чтобы сумма всех элементов была равна 1.
* Размер ядра часто выбирается нечетным (например, 3x3, 5x5, 7x7).

**4. Алгоритм размытия по Гауссу**

Алгоритм размытия по Гауссу включает следующие шаги:

1. Определить размер ядра и значение стандартного отклонения (sigma).
2. Создать матрицу Гаусса на основе заданного размера и sigma.
3. Применить свертку к изображению, используя созданное ядро Гаусса.
4. Для каждого пикселя изображения:
   * Учитывать значения пикселей в области вокруг него в соответствии с весами из ядра.
   * Вычислить новое значение пикселя, суммируя взвешенные значения.

**5. Параметры размытия по Гауссу**

Некоторые параметры размытия по Гауссу включают:

* **Размер ядра**: Определяет, сколько пикселей вокруг центрального пикселя будет учитываться.
* **Стандартное отклонение (sigma)**: Указывает, насколько "широким" будет размытие. Большие значения sigma приводят к более сильному размытия.
* **Граница обработки**: Как обрабатывать пиксели на краях изображения (например, отразить или заполнить).

**6. Фильтрация по Гауссу с использованием OpenCV**

В OpenCV фильтрация по Гауссу выполняется с помощью функции **cv2.GaussianBlur()**.