

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ»  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)  
Институт №8 «Компьютерные науки и прикладная математика»  
Кафедра 806: «Вычислительная математика и программирование»

# ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ №3

По курсу: «Цифровая обработка изображений»

Тема: «Фильтрация изображений в пространственной области»

*Студент:* Чернышев Д.В.

*Группа:* М8О-107М-22

*Вариант:* 8

*Преподаватель:* Гаврилов К.Ю.

Москва

2023

## **СОДЕРЖАНИЕ**

<b>1</b>	<b>Задание к лабораторной работе №3.....</b>	<b>3</b>
<b>2</b>	<b>Выполнение лабораторной работы №3 .....</b>	<b>5</b>
<b>3</b>	<b>Выводы .....</b>	<b>19</b>

# 1 Задание к лабораторной работе №3

Дано изображение *Img3\_08\_1.jpg*.

## Часть 1

Исследование пространственных фильтров для подавления шумов на изображениях

1. Моделирование искажения изображений с помощью шумов. Проведите моделирование искажения изображения путем добавления следующих шумов: гауссовский шум, шум типа «соль и перец».
2. Составьте программу, выполняющую подавление шумов с помощью масочных фильтров следующего вида:
  - среднеарифметический;
  - среднегеометрический;
  - среднегармонический
  - медианный.
3. Проведите исследование эффективности подавления шумов обоих видов (гауссовского и «соль и перец») при различных уровнях зашумления и при использовании фильтров различного вида.

## Часть 2

Использование пространственных фильтров для повышения резкости изображения.

1. Использование Лапласиана. Используя расфокусировку с помощью гауссовского фильтра получите слегка размытое изображение. Используя Лапласиан улучшите резкость изображения. Путем анализа некоторых деталей изображения (букв, символов, мелких деталей) сделайте выводы о степени улучшения изображения в результате использования Лапласиана (например, стали ли читаться буквы, насколько лучше видны отдельные детали и т.д.). Используйте две функции Matlab: *imfilter* с маской Лапласиана и *locallapfilt* (в

моем случае аналоги в *Python*). Сравните результаты улучшения изображения.

2. Использование маски Собела. Используя такую же расфокусировку заданного изображения, что и в п.1, улучшите резкость с помощью масочного фильтра Собела.
3. Сравните эффективность работы двух алгоритмов улучшения резкости изображения: Лапласиана и оператора Собела.



Рис. 1: Исходное изображения для обработки

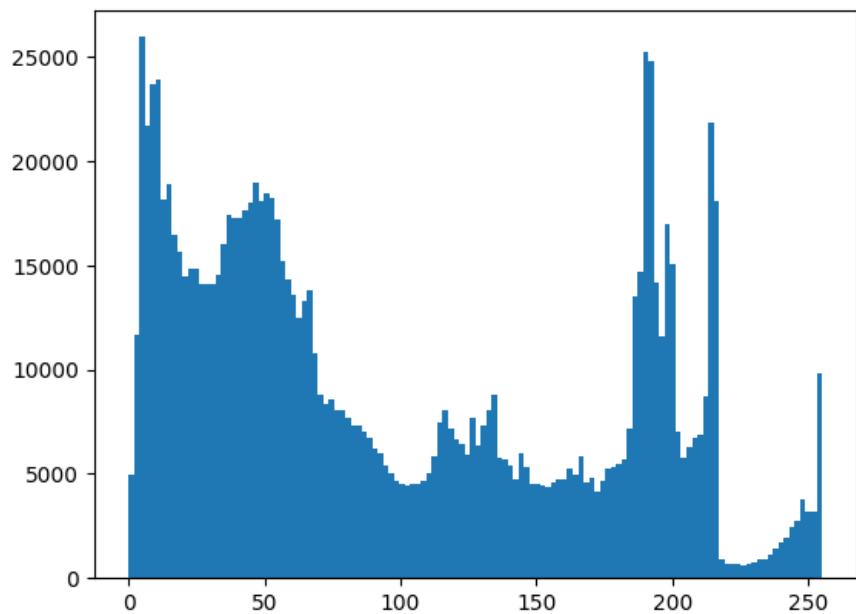


Рис. 2: Исходное изображения для обработки (гистограмма)

## 2 Выполнение лабораторной работы №3

### Часть 1

Производим импорт необходимых библиотек которые нам понадобятся

```
1 import numpy as np
2 import PIL
3 import os
4 import cv2
5 import bisect
6 import matplotlib.image as mpimg
7 from skimage.util import random_noise
8 from scipy.ndimage import gaussian_filter
9 from numba import jit
10
11 from PIL import Image, ImageDraw, ImageStat
12 import random
13 import matplotlib.pyplot as plt
14
15 # from scipy import ndimage, datasets
16 import scipy
17 import scipy.stats
18 import random
19
```

Далее определим функции зашумления фотографий

```
1 def sp_noise(image,prob=float(0.05)):
2     """
3         Add salt and pepper noise to image
4         prob: Probability of the noise
5     """
6     assert 0 <= prob <= 1, "Probability of the noise
7         must be less
```

```

8     or equal to 1 or more or equal to 0"
9     output = np.zeros(image.shape,np.uint8)
10    thres = 1 - prob
11    for i in range(image.shape[0]):
12        for j in range(image.shape[1]):
13            rdn = random.random()
14            if rdn < prob:
15                output[i][j] = 0
16            elif rdn > thres:
17                output[i][j] = 255
18            else:
19                output[i][j] = image[i][j]
20
21
22 def noisy(
23     noise_typ:str,
24     image:np.ndarray,
25     prob:float=0.05) -> np.ndarray:
26     """_summary_
27     Args:
28         noise_typ : str - One of the following strings,
29         selecting the type of noise to add:
30             (*) 'gauss'      Gaussian-distributed additive noise.
31             (*) 'poisson'    Poisson-distributed
32             noise generated from the data.
33             (*) 's_p'        Replaces random pixels
34             with 0 or 1.
35             (*) 'speckle'    Multiplicative noise using
36             out = image + n*image, where n is uniform noise
37             with specified mean & variance.
38             image : ndarray - Input image data.
39             Will be converted to float.
40
41     Returns:

```

```

41         noisy: ndarray - noised image
42         """
43
44     assert 0 <= prob <= 1, "Probability of the noise
45     must be less or equal to 1 or more or equal to 0"
46     if noise_typ == "gauss":
47         row,col,ch= image.shape
48
49         mean = 255/2
50
51         var = 0.1
52
53         sigma = var ** 0.5
54
55         gauss = np.random.normal(mean,sigma,(row,col,ch))
56         gauss = gauss.reshape(row,col,ch)
57         noisy = image + gauss
58         return noisy
59
60
61     elif noise_typ == "s_p":
62         return sp_noise(image,prob)
63
64
65     elif noise_typ == "poisson":
66         vals = len(np.unique(image))
67         vals = 2 ** np.ceil(np.log2(vals))
68         noisy = np.random.poisson(image * vals) / float(vals)
69         return noisy
70
71
72     elif noise_typ =="speckle":
73         row,col,ch = image.shape
74         gauss = np.random.randn(row,col,ch)
75         gauss = gauss.reshape(row,col,ch)
76         noisy = image + image * gauss
77         return noisy

```

Применим гауссовский шум и шум типа "соль и перец" к исходному изображения

Получим:



Рис. 3: Применение гауссовского шума

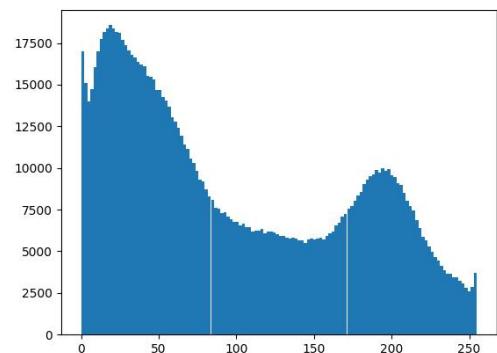


Рис. 4: Применение гауссовского шума (гистограмма)



Рис. 5: Применение S&P шума

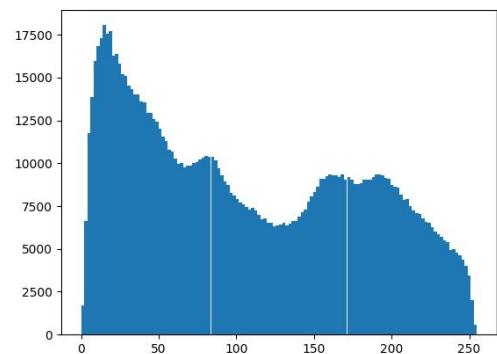


Рис. 6: Применение S&P шума (гистограмма)

Необходимо определить функции фильтров

```
1 def GaussieNoisy(image,sigma):
2     #
3     img = image.astype(np.int16) # Этот шаг - избежать случая,
4     # когда точка пикселей меньше 0, превышает 255
5     mu = 0
6     for i in range(img.shape[0]):
7         for j in range(img.shape[1]):
8             for k in range(img.shape[2]):
9                 img[i, j, k] = img[i, j, k] + random.gauss(
10                     mu=mu,
11                     sigma=sigma
12                 )
13     img[img > 255] = 255
14     img[img < 0] = 0
15     img = img.astype(np.uint8)
16     return img
17
18 def spNoisy(image,s_vs_p = 0.5,amount = 0.004):
19     #
20     out = np.copy(image)
21     num_salt = np.ceil(amount * image.size * s_vs_p)
22     coords = [np.random.randint(
23         0,
24         i - 1,
25         int(num_salt)
26     ) for i in image.shape]
27     out[tuple(coords)] = 1
28     num_pepper = np.ceil(amount* image.size * (1. - s_vs_p))
29     coords = [np.random.randint(
30         0,
31         i - 1,
32         int(num_pepper)
```

```

33     ) for i in image.shape]
34     out[tuple(coords)] = 0
35
36
37
38 def ArithmeticMeanAlogrithm(image):
39     # Арифметический средний фильтр
40     new_image = np.zeros(image.shape)
41     image = cv2.copyMakeBorder(
42         image,
43         1,1,1,1,
44         cv2.BORDER_DEFAULT
45     )
46     for i in range(1,image.shape[0]-1):
47         for j in range(1,image.shape[1]-1):
48             new_image[i-1,j-1] = np.mean(image[i-1:i+2,j-1:j+2])
49     new_image = (new_image-np.min(image))*(255/np.max(image))
50     return new_image.astype(np.uint8)
51
52 def rgbArithmeticMean(image):
53     r,g,b = cv2.split(image)
54     r = ArithmeticMeanAlogrithm(r)
55     g = ArithmeticMeanAlogrithm(g)
56     b = ArithmeticMeanAlogrithm(b)
57     return cv2.merge([r,g,b])
58
59
60 def GeometricMeanOperator(roi):
61     roi = roi.astype(np.float64)
62     p = np.prod(roi)
63     return p ** (1 / (roi.shape[0] * roi.shape[1]))
64
65 def GeometricMeanAlogrithm(image):
66     # Геометрическая средняя фильтрация
67     new_image = np.zeros(image.shape)

```

```

66     image = cv2.copyMakeBorder(
67         image,
68         1,1,1,1,
69         cv2.BORDER_DEFAULT
70     )
71     for i in range(1, image.shape[0] - 1):
72         for j in range(1, image.shape[1] - 1):
73             new_image[i - 1, j - 1] = GeometricMeanOperator(
74                 image[i - 1:i + 2, j - 1:j + 2
75             ])
76     new_image = (new_image - np.min(image))
77             * (255 / np.max(image))
78     return new_image.astype(np.uint8)
79 def rgbGeometricMean(image):
80     r,g,b = cv2.split(image)
81     r = GeometricMeanAlgorithm(r)
82     g = GeometricMeanAlgorithm(g)
83     b = GeometricMeanAlgorithm(b)
84     return cv2.merge([r,g,b])
85
86 def HarmonicMeanOperator(roi):
87     roi = roi.astype(np.float64)
88     if 0 in roi:
89         roi = 0
90     else:
91         roi = scipy.stats.hmean(roi.reshape(-1))
92     return roi
93 def HarmonicMeanAlgorithm(image):
94     # Гармонический средний фильтр
95     new_image = np.zeros(image.shape)
96     image = cv2.copyMakeBorder(
97         image,
98         1,1,1,1,

```

```

99         cv2.BORDER_DEFAULT
100    )
101    for i in range(1,image.shape[0]-1):
102        for j in range(1,image.shape[1]-1):
103            new_image[i-1,j-1] =HarmonicMeanOperator(
104                image[i-1:i+2,j-1:j+2]
105            )
106    new_image = (new_image-np.min(image))*(255/np.max(image))
107    return new_image.astype(np.uint8)
108
109 def rgbHarmonicMean(image):
110     r,g,b = cv2.split(image)
111     r = HarmonicMeanAlogrithm(r)
112     g = HarmonicMeanAlogrithm(g)
113     b = HarmonicMeanAlogrithm(b)
114     return cv2.merge([r,g,b])
115
116
117 def Contra_harmonicMeanOperator(roi,q):
118     roi = roi.astype(np.float64)
119     return np.mean((roi)**(q+1))/np.mean((roi)**(q))
120
121 def Contra_harmonicMeanAlogrithm(image,q):
122     # Обратная гармонический средний фильтр
123     new_image = np.zeros(image.shape)
124     image = cv2.copyMakeBorder(
125         image,
126         1,1,1,1,
127         cv2.BORDER_DEFAULT)
128     for i in range(1,image.shape[0]-1):
129         for j in range(1,image.shape[1]-1):
130             new_image[i-1,j-1] = Contra_harmonicMeanOperator(
131                 image[i-1:i+2,j-1:j+2],
132                 q
133             )

```

```
132     new_image = (new_image-np.min(image))*(255/np.max(image))
133     return new_image.astype(np.uint8)
134 def rgbContra_harmonicMean(image,q):
135     r,g,b = cv2.split(image)
136     r = Contra_harmonicMeanAlogrithm(r,q)
137     g = Contra_harmonicMeanAlogrithm(g,q)
138     b = Contra_harmonicMeanAlogrithm(b,q)
139     return cv2.merge([r,g,b])
```

Применяем фильтры к двум зашумленным изображениям

## S&P

```
1 # Загрузка изображения для фильтров
2 sp_image = cv2.imread(SP_PATH_FILE)
3 # Среднегеометрический фильтр
4 noise_img_sp_GeomFilter = rgbGemotriccMean(sp_image)
5 # Среднееарифметический фильтр
6 noise_img_sp_AirthmFilter = rgbArithmeticMean(sp_image)
7 # Среднегармонический фильтр
8 noise_img_sp_HarmonicFilter = rgbHarmonicMean(sp_image)
9 # Медианный фильтр
10 noise_img_sp_MedianFilter = scipy.ndimage.median_filter(
11     sp_image,
12     size=20
13 )
```

## Gauss

```
1 # Загрузка изображения для фильтров
2 gauss_image = cv2.imread(GAUSS_PATH_FILE)
3 # Среднегеометрический фильтр
4 noise_img_gauss_GeomFilter = rgbGemotriccMean(gauss_image)
5 # Среднееарифметический фильтр
6 noise_img_gauss_AirthmFilter = rgbArithmeticMean(gauss_image)
7 # Среднегармонический фильтр
8 noise_img_gauss_HarmonicFilter = rgbHarmonicMean(gauss_image)
9 # Медианный фильтр
10 noise_img_gauss_MedianFilter = scipy.ndimage.median_filter(
11     gauss_image,
12     size=20
13 )
```

Таблица 1: Фильтрация S&P-зашумленного изображения разными фильтрами

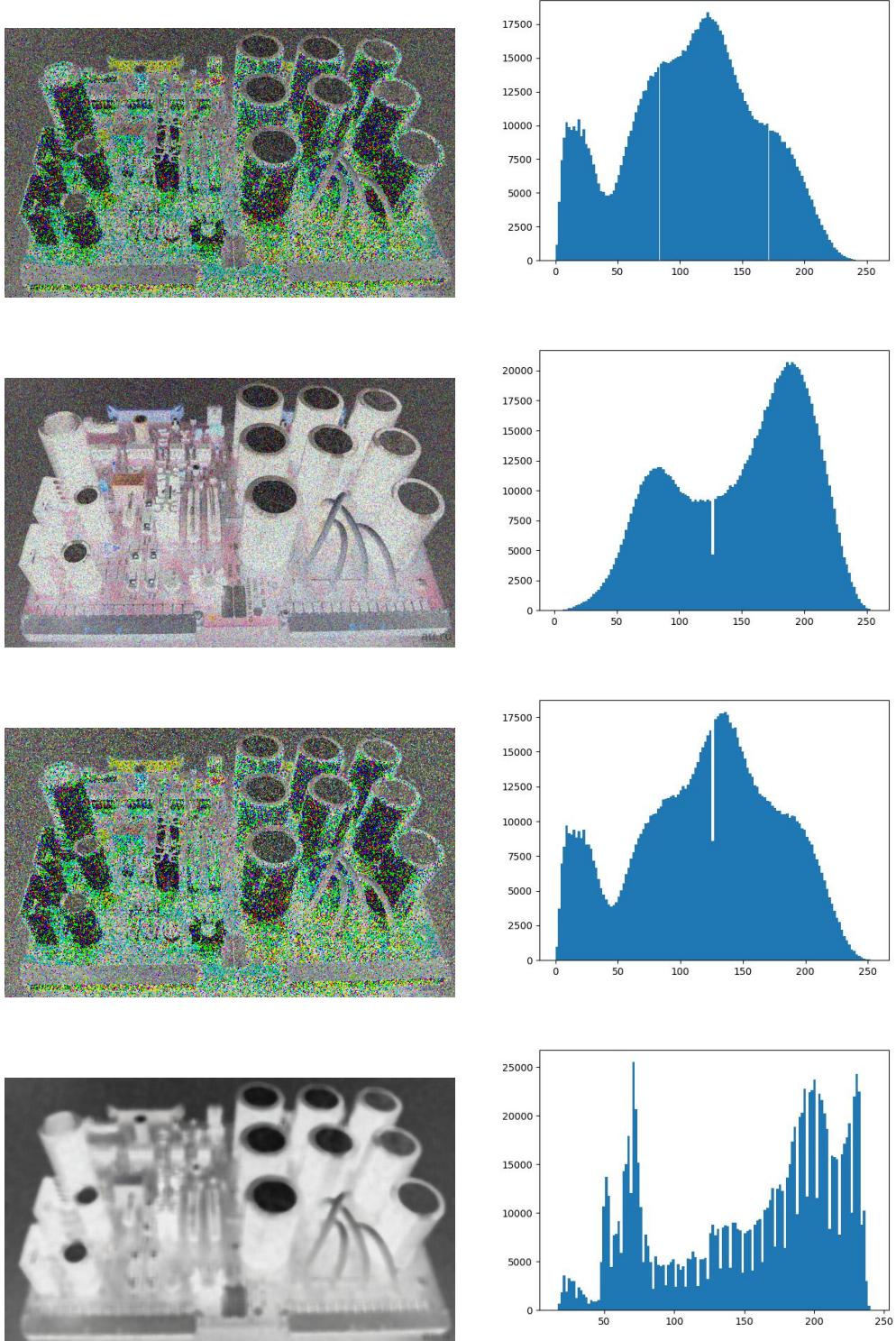
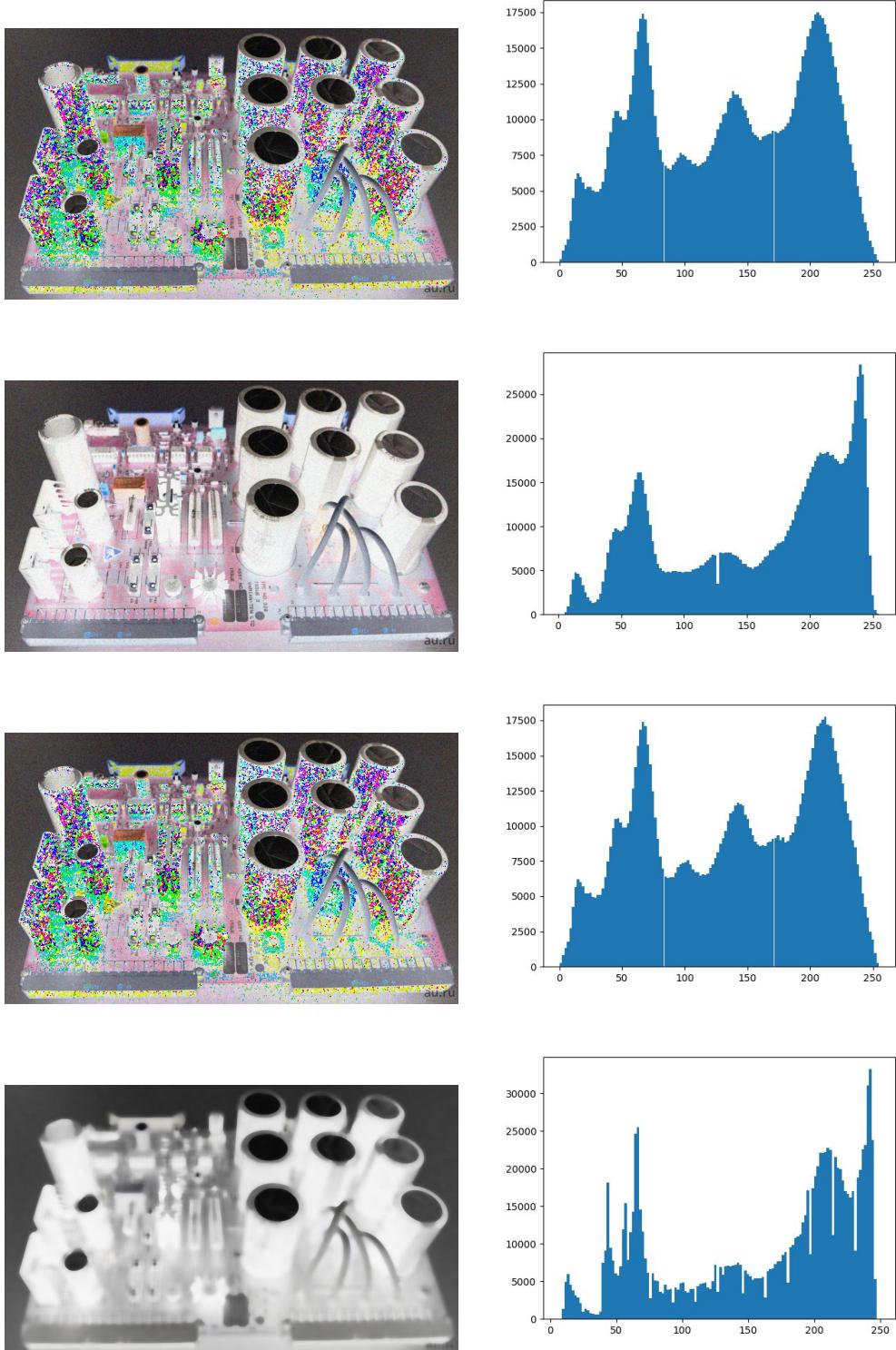


Таблица 2: Фильтрация гауссово-зашумленного изображения разными фильтрами



## Часть 2

Применяем гауссовский фильтра для расфокусировки:

```
1 img = cv2.imread(SOURCE_PIC)
2 noise_img_sp = gaussian_filter(img, sigma=3)
3 im = Image.fromarray((noise_img_sp * 255).astype(np.uint8))
4 Gauss_PATH_FILE = os.path.join(
5     OUTPUT_DIR_2 ,
6     f 'Gauss_{SOURCE_PIC}'
7 )
8 im.save(Gauss_PATH_FILE)
9 plt.imshow(noise_img_sp)
```

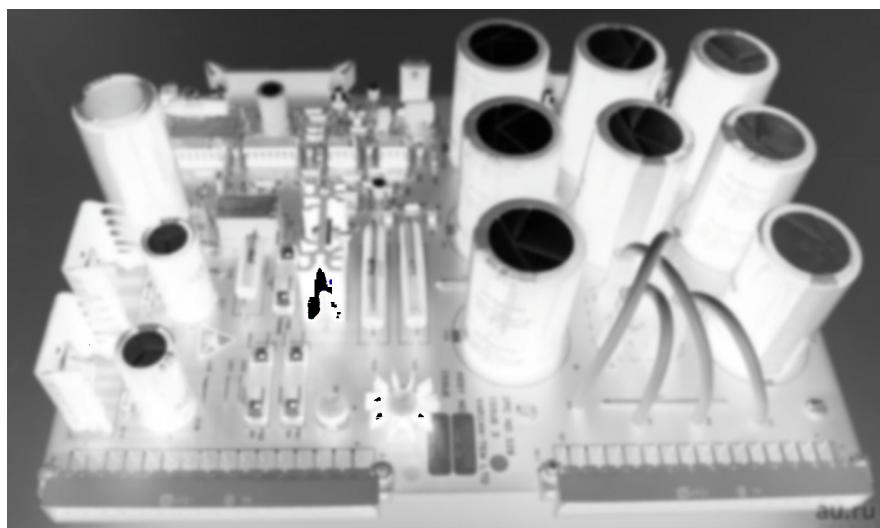


Рис. 7: Расфокусированное изображение

Применяем

```
1 # convolute with proper kernels
2 laplacian = cv2.Laplacian(noise_img_sp, cv2.CV_64F)
3 sobelx = cv2.Sobel(noise_img_sp, cv2.CV_64F, 1, 0, ksize=3) # x
4 sobely = cv2.Sobel(noise_img_sp, cv2.CV_64F, 0, 1, ksize=3) # y
5 sobelxy = cv2.Sobel(noise_img_sp, cv2.CV_64F, 1, 1, ksize=3) # xy
```

Результаты:

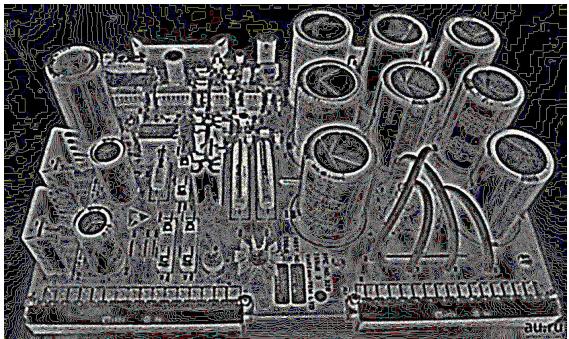


Рис. 8: Применение Лапласиана

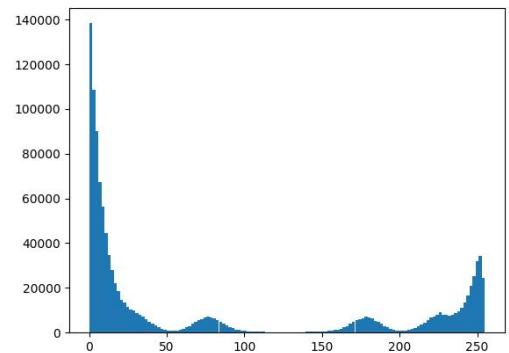


Рис. 9: Применение Лапласиана  
(гистограмма)

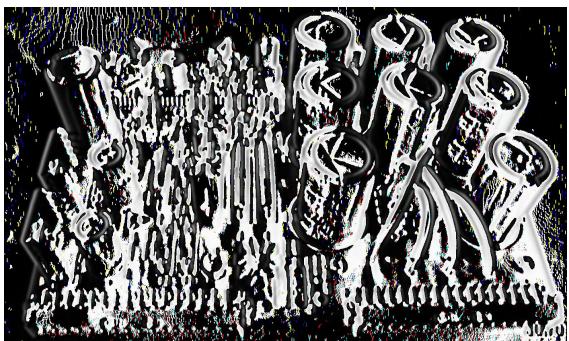


Рис. 10: Применение *Sobel X*

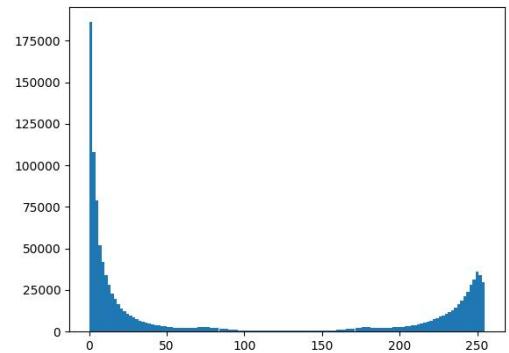


Рис. 11: Применение *Sobel X*  
(гистограмма)

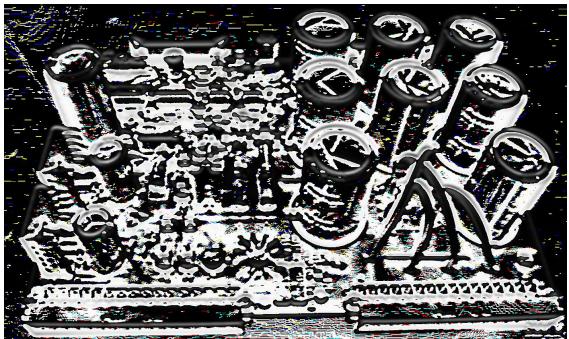


Рис. 12: Применение *Sobel Y*

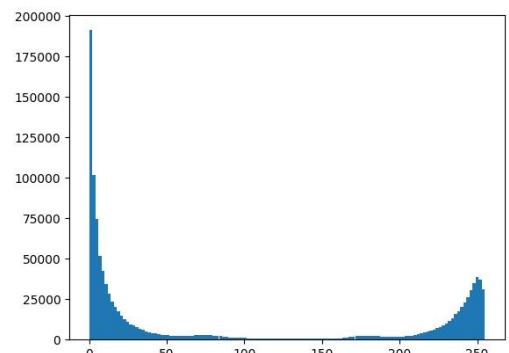


Рис. 13: Применение *Sobel Y*  
(гистограмма)

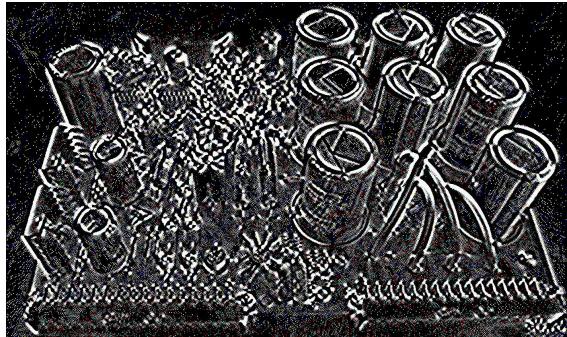


Рис. 14: Применение *Sobel XY*

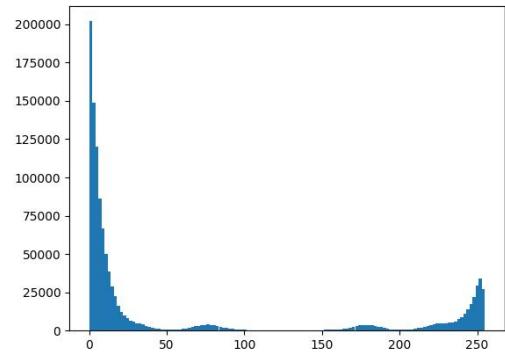


Рис. 15: Применение *Sobel XY*  
(гистограмма)

### 3 Выводы

По результатам выполнения лабораторной работы №3 было продемонстрировано действие различных фильтров на зашумленные изображения, получены и проанализированы диаграммы исходных и отфильтрованных изображений.