

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное  
учреждение высшего образования  
«Самарский национальный исследовательский университет  
имени академика С.П. Королева»  
Институт информатики и кибернетики  
Кафедра технической кибернетики

**Отчет по лабораторной работе № 1**  
**«Поэлементные преобразования изображений»**

Курс: «Системы обработки изображений»

Студент: Грязнов Илья Евгеньевич  
(фамилия, имя отчество)

Группы 6131-010402D

**Самара 2022**

## Содержание

Задание .....	3
Ход выполнения работы: .....	6
1. Считать цветное rgb изображение .....	6
2. Преобразовать изображение в градации серого .....	7
3. Написать функцию реализации препарирования изображения .....	9
4. По гистограмме входного изображения определить значение порога яркостей, обеспечивающего оптимальное разделение объекта и фона. Осуществить пороговую обработку входного изображения с найденным пороговым значением. ....	10
5. Сделать пороговую обработку методом Otsu (Функция OpenCV) .....	15
6. Определить динамический диапазон входного изображения. Осуществить линейное контрастирование входного изображения в заданный динамический диапазон яркостей .....	16
7. Сделать эквализацию гистограммы изображения .....	19
8. Сделать эквализацию методом CLANE (Функция OpenCV) .....	20
9. Осуществить препарирование изображения с заданной препарирующей функцией .....	21
ЗАКЛЮЧЕНИЕ .....	25

Задание  
«Поэлементные преобразования изображений»

**Вариант № 6**

1. Считать цветное rgb изображение
2. Преобразовать изображение в градации серого
3. Написать функцию реализации поэлементной обработки изображения

*Функцию вида  $fun(Image, prepfun)$*

*Где  $prepfun$  - конкретная функция препаирования, заданная в том или ином виде*

4. По гистограмме изображения определить значение порога яркостей, обеспечивающего оптимальное разделение объекта и фона. Осуществить пороговую обработку входного изображения с найденным пороговым значением

- \* Вход: изображение из пункта 2

- \* Вывод: входное и результирующие изображение и их гистограммы соответственно

- \* Порог определить на глаз по гистограмме

5. Сделать пороговую обработку методом Otsu (Функция OpenCV)

- \* Вход: изображение из пункта 2

- \* Вывод: входное и результирующие изображение и их гистограммы соответственно

6. Определить динамический диапазон входного изображения. Осуществить линейное контрастирование входного изображения в заданный динамический диапазон яркостей

\* Вход: изображение у которого диапазон не на всем промежутке [0,255] или использовать:



\* Вывод: входное и результирующие изображение и их гистограммы соответственно

\* Вычислить коэффициенты  $a$  и  $b$ . Сделать преобразование вида  $g = a * f + b$

7. Сделать эквализацию гистограммы изображения

\* Вход: изображение из пункта 2

\* Вывод: входное и результирующие изображение и их гистограммы соответственно

8. Сделать эквализацию методом CLAHE (Функция OpenCV)

\* Вход: изображение из пункта 2

\* Вывод: входное и результирующие изображение и их гистограммы соответственно


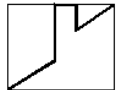
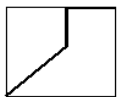


9. Осуществить препарирование изображения с заданной препарирующей функцией


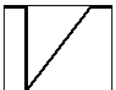
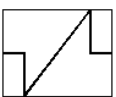
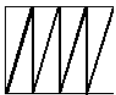

\* Вход: изображение из пункта 2

\* Вывод: входное и результирующие изображение и их гистограммы соответственно. График препарирующий функции.

\* Пороги в препарирующей функции выбирать самостоятельно

#### Варианты препарирующей функции

№ варианта	Вид функции
1	
2	
3	
4	
5	

№ варианта	Вид функции
6	
7	
8	
9	
10	

## Ход выполнения работы:

### 1. Считать цветное rgb изображение

С помощью библиотеки CV2 было подгружено два изображения, так же отображена размерность каждого изображения.

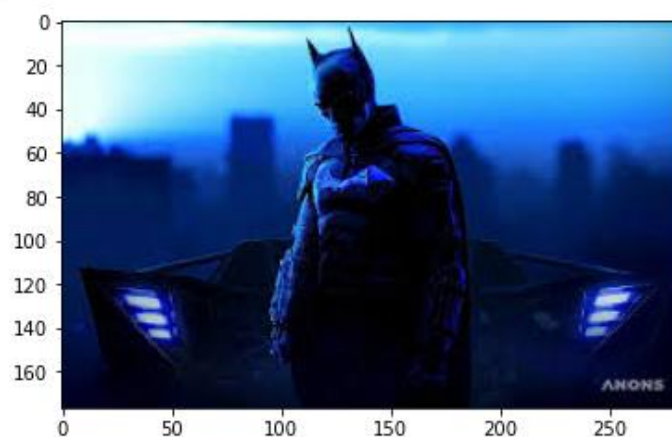
```
In [3]: image = cv2.imread("img.jpg")  
plt.imshow(image)  
plt.show()
```



```
In [4]: image.shape
```

```
Out[4]: (1194, 2560, 3)
```

```
In [5]: image1 = cv2.imread("images1.jpg")  
plt.imshow(image1)  
plt.show()
```



```
In [6]: image1.shape
```

```
Out[6]: (177, 284, 3)
```

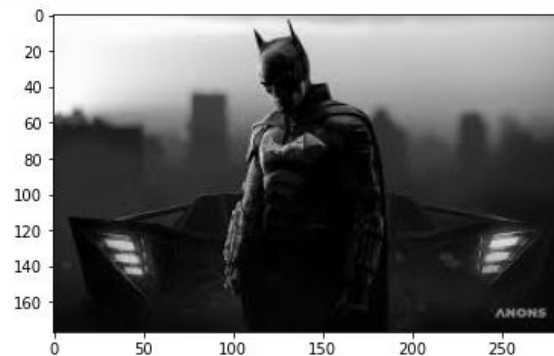
Для отображения самих изображений использовалась библиотека «matplotlib».

## 2. Преобразовать изображение в градации серого

Для преобразования использовались разные методы.

```
In [7]: # Библиотекой  
img = Image.open("images1.jpg").convert('LA')
```

```
In [8]: plt.imshow(img)  
plt.show()
```

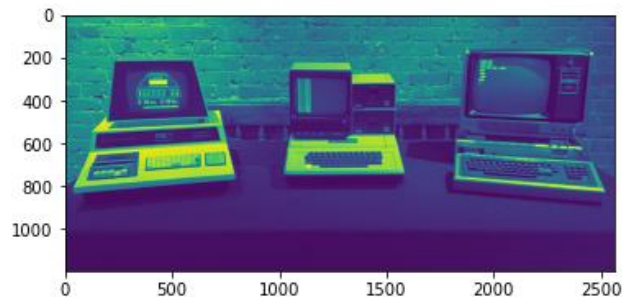


При помощи библиотеки:

И реализовано преобразование самостоятельно методом средних значений. Средний метод является наиболее простым. Просто нужно взять в среднем три цвета. Так как это RGB-изображение, это означает, что необходимо добавить R с G к B, а затем разделить на 3, чтобы получить желаемое изображение в градациях серого.

```
In [9]: # Самостоятельно  
  
# Среднее значение  
img_f = img_as_float(image)  
  
avg_gray = (img_f[:, :, 0] + img_f[:, :, 1] + img_f[:, :, 2]) / 3  
plt.imshow(avg_gray)
```

```
Out[9]: <matplotlib.image.AxesImage at 0x25e4aa5bbe0>
```



```
In [10]: avg_gray.shape
```

```
Out[10]: (1194, 2560)
```

Это сделано таким образом. Оттенки серого =  $(R + G + B / 3)$   
 Далее был использован взвешенный метод или метод светимости. Проблема, которая возникает в среднем метода – смещение цветов в темную сторону. Взвешенный метод имеет решение этой проблемы. Поскольку красный цвет имеет большую длину волны из всех трех цветов, а зеленый – это цвет, который имеет не только меньшую длину волны, чем красный, но и зеленый – это цвет, который дает более успокаивающий эффект для глаз.

Это означает, что мы должны уменьшить вклад красного цвета, увеличить вклад зеленого цвета и поместить вклад синего цвета между этими двумя.

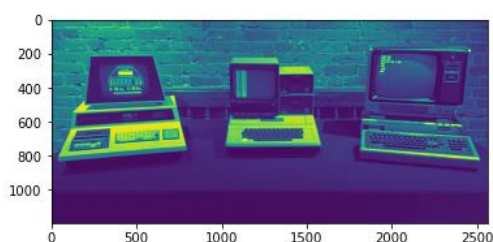
Итак, новое уравнение этой формы:

Новое изображение в градациях серого =  $((0,21 * R) + (0,71 * G) + (0,07 * B))$ .

Согласно этому уравнению, красный дает ~21%, зеленый ~ 71%, что больше во всех трех цветах, а синий ~ 7%.

```
In [11]: img_gray = (img_f[:, :, 0] * 0.2126 + img_f[:, :, 1] * 0.7152 + img_f[:, :, 2] * 0.0722)
plt.imshow(img_gray)
```

```
Out[11]: <matplotlib.image.AxesImage at 0x25e4aad0730>
```



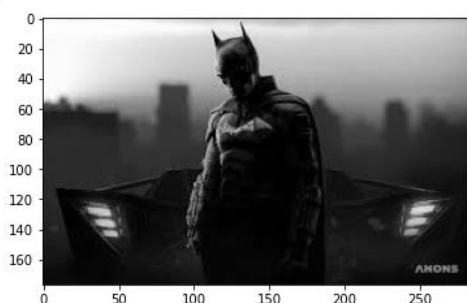
```
In [12]: img_gray = img_as_ubyte(img_gray)

plt.figure(figsize = (20,20))

plt.subplot(1, 3, 1)

plt.imshow(img, cmap = plt.cm.gist_gray)
plt.subplot(1, 3, 2)

plt.imshow(img_gray, cmap=plt.cm.gist_gray)
plt.show()
```





### 3. Написать функцию реализации препарирования изображения

Препарирование используется, когда необходимо подчеркнуть, усилить какие-то черты, особенности, нюансы наблюдаемого изображения с целью улучшения субъективного восприятия. На результат оказывает влияние значение интенсивности только в обрабатываемой точке, а не в ее окрестности.

Препарирование используется, когда необходимо подчеркнуть, усилить какие-то черты, особенности, нюансы наблюдаемого изображения с целью улучшения субъективного восприятия.

```
In [13]: # Функция совершает препарирование картинки

def prepare_OLD(input_image, dot1, dot2): # просто оставил старый вариант
    output_image = input_image.copy()
    output_image[output_image <= dot1] = 0
    output_image[output_image >= dot2] = 0
    output_image = np.where((output_image > dot1) & (output_image < dot2), ((output_image-dot1)/(dot2-dot1)*255).astype(int), output_image)
    return output_image

In [14]: def prepare_func_v3(x):
    dot1=80
    dot2=230
    if x <= dot1:
        return 0
    elif x >= dot2:
        return 0
    else:
        # return x
        return int((x-dot1)/(dot2-dot1)*255)

In [15]: def prepare_v3(input_image, func):
    new_function = np.vectorize(func)
    return new_function(input_image)
```

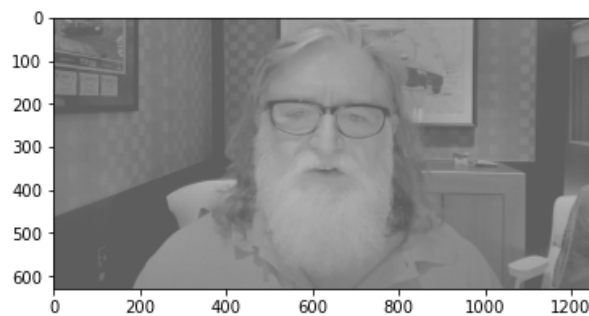
Пусть  $x(i, j) = x_{ij}$ ,  $y(i, j) = y_{ij}$  – значения яркости исходного и получаемого после обработки изображений соответственно в точке кадра, имеющей декартовы координаты  $i$  (номер строки) и  $j$  (номер столбца). Поэлементная обработка означает, что существует функциональная однозначная зависимость  $y_{ij} = f_{ij}(x_{ij})$  между этими яркостями одинаковая для всех точек кадра.

**4. По гистограмме входного изображения определить значение порога яркостей, обеспечивающего оптимальное разделение объекта и фона. Осуществить пороговую обработку входного изображения с найденным пороговым значением.**

Некоторые задачи обработки видеоинформации связаны с преобразованием полутонового изображения (т.е. такого, которое имеет много градаций яркости) в бинарное (двухградационное). Такое преобразование осуществляется в первую очередь для того, чтобы сократить информационную избыточность

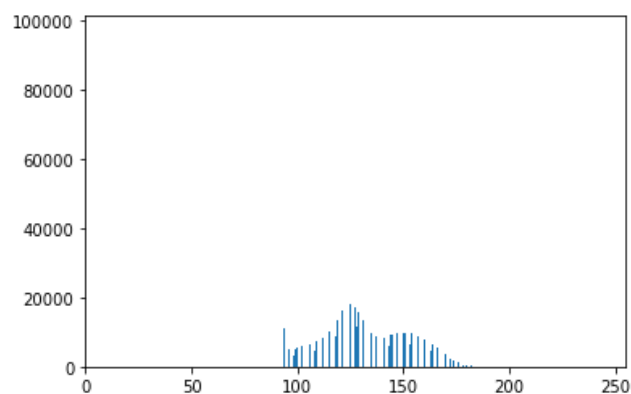
```
In [16]: im = cv2.imread("index.png")
plt.imshow(im)

Out[16]: <matplotlib.image.AxesImage at 0x25e4ac01bb0>
```



```
In [17]: # Расчет среднего значения из каналов RGB и сгладить до массива 1D
vals = im.mean(axis=2).flatten()

# построение гистограммы с 255 интервалами
b, bins, patches = plt.hist(vals, 255)
plt.xlim([0,255])
plt.show()
```

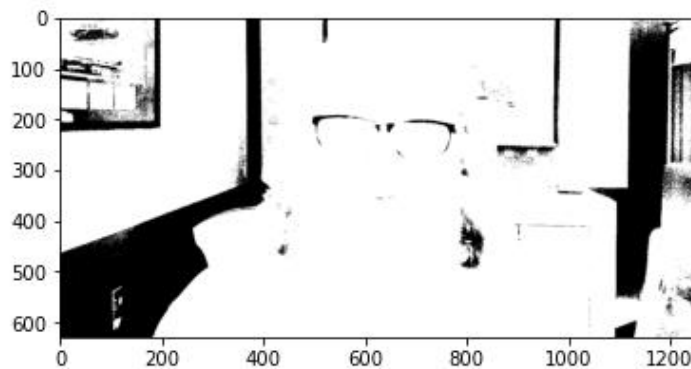


изображения, оставив в нем только ту информацию, которая нужна для решения конкретной задачи. В бинарном изображении (или, как говорят, графическом

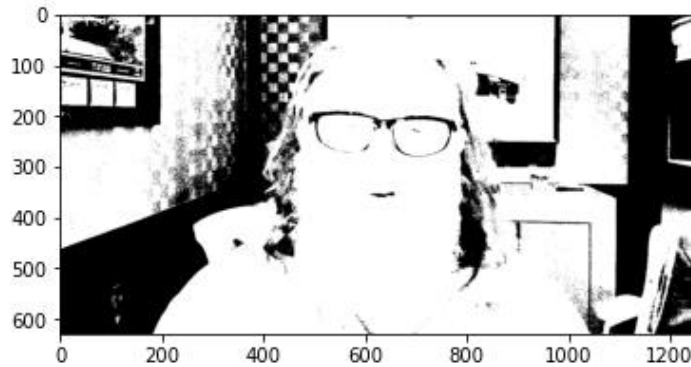
препарате полутонового) должны быть сохранены интересующие нас детали (например, очертания изображенных объектов и исключены несущественные особенности (фон)).

```
In [18]: for threshold_value in range (100,250,15):  
  
    print("threshold_value =", threshold_value)  
    plt.figure()  
    img_threshold = np.where(np.array(im)<=threshold_value, 0, 255)  
    plt.imshow(img_threshold)  
    plt.show()
```

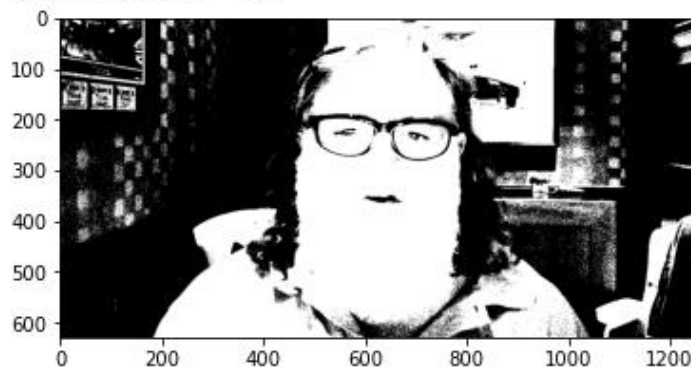
threshold\_value = 100



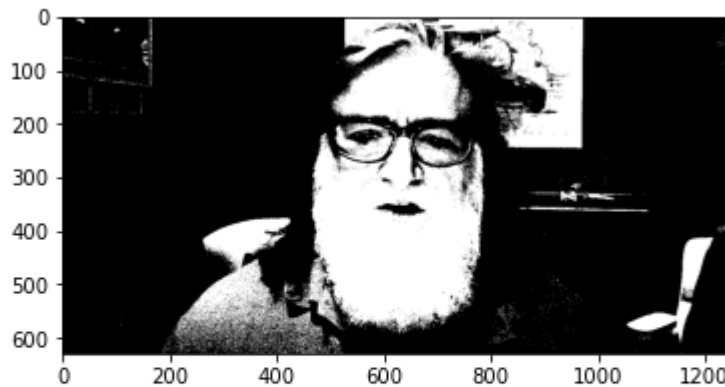
threshold\_value = 115



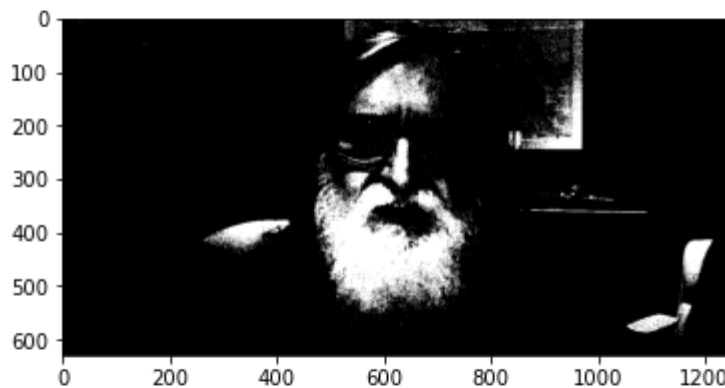
threshold\_value = 130



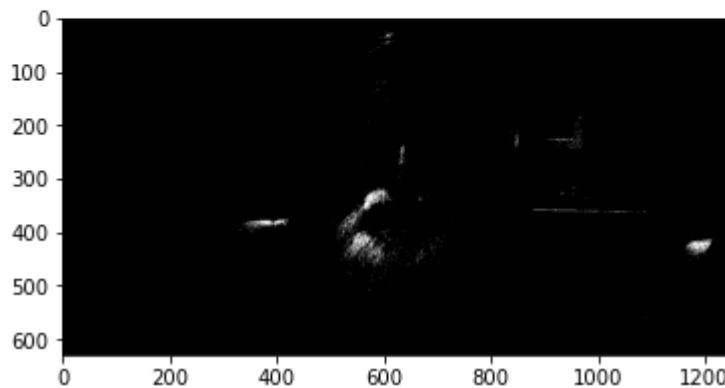
```
threshold_value = 145
```



```
threshold_value = 160
```



```
threshold_value = 175
```

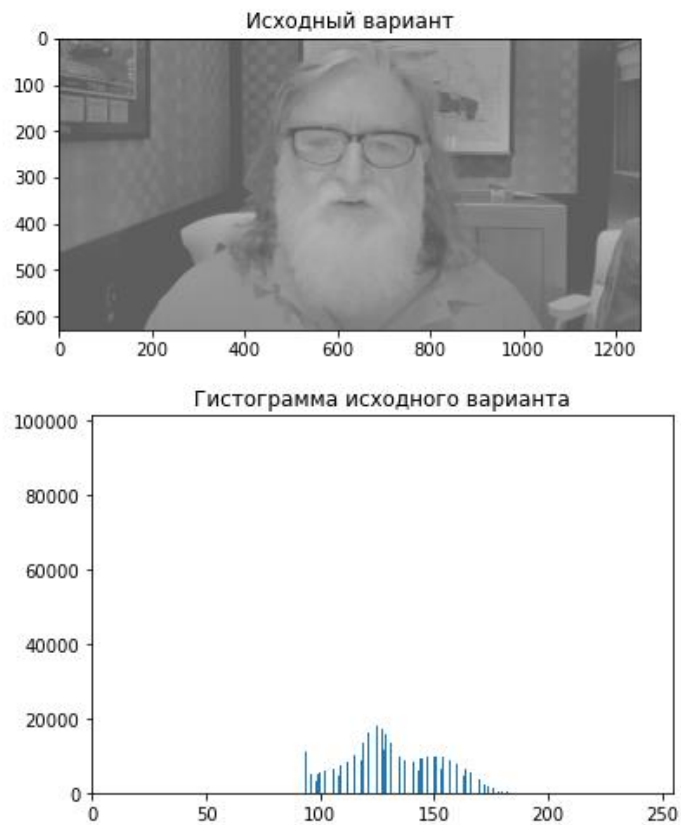


**Вывод:** По гистограмме изображения определил два "пика" яркостей, предположив эти значения как пороги яркостей, обеспечивающего оптимальное разделение объекта и фона. Эти значения примерно 120 и 150. Далее императивным путем провел проверку и визуально увидел, что наиболее приемлемый порог в `threshold_value = 140`

Ниже представлены входное и результирующее изображение и их гистограммы соответственно:

```
In [19]: plt.figure()
plt.title("Исходный вариант")
plt.imshow(im)

plt.figure()
plt.title("Гистограмма исходного варианта")
# Расчет среднего значения из каналов RGB и сгладить до массива 1D
vals = im.mean(axis=2).flatten()
# построение гистограммы с 255 интервалами
plt.hist(vals, 255)
plt.xlim([0,255])
plt.show()
```



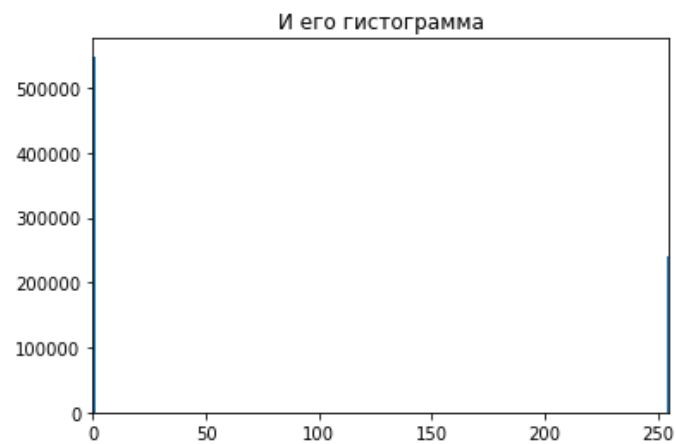
```
In [20]: # Наилучший результат достигается при threshold_value = 140
threshold_value = 140
plt.figure()
img_threshold = np.where(np.array(im)<=threshold_value, 0, 255)

plt.figure()
plt.title("Выбранный мной эмпирическим путем вариант")

plt.imshow(img_threshold)
plt.show()

plt.figure()
plt.title("И его гистограмма")
# Расчет среднего значения из каналов RGB и сгладить до массива 1D
vals = img_threshold.mean(axis=2).flatten()
# построение гистограммы с 255 интервалами
plt.hist(vals, 255)
plt.xlim([0,255])
plt.show()
```

<Figure size 432x288 with 0 Axes>



## 5. Сделать пороговую обработку методом Otsu (Функция OpenCV)

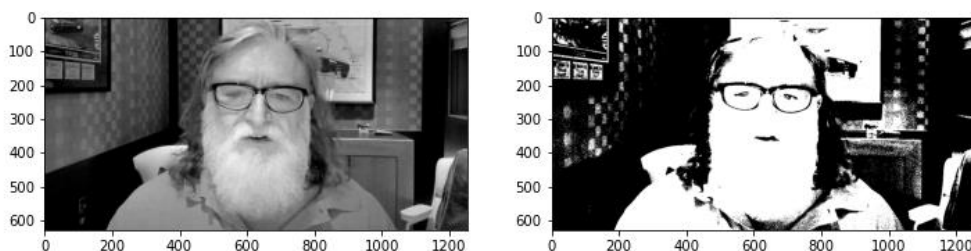
Так же пороговую обработку изображения позволяет реализовать библиотека OpenCV.

OpenCV предназначена для повышения вычислительной эффективности процедур обработки видеоизображения с особым упором на применение в задачах реального времени. OpenCV написана на C хорошо оптимизирована и может использовать преимущества многоядерных процессоров. Для более полного использования возможностей библиотеки рекомендуется установить Intel Performance Primitives (IPP). Это позволит повысить производительность процедур библиотеки.

```
In [21]: img_gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
# Заново в режиме серого

ret, th1 = cv2.threshold(img_gray, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
plt.figure(figsize=(12,10))
plt.subplot(221),plt.imshow(img_gray,'gray')
plt.subplot(222),plt.imshow(th1,'gray')
```

```
Out[21]: (<AxesSubplot:~>, <matplotlib.image.AxesImage at 0x25e4b3afaf0>)
```

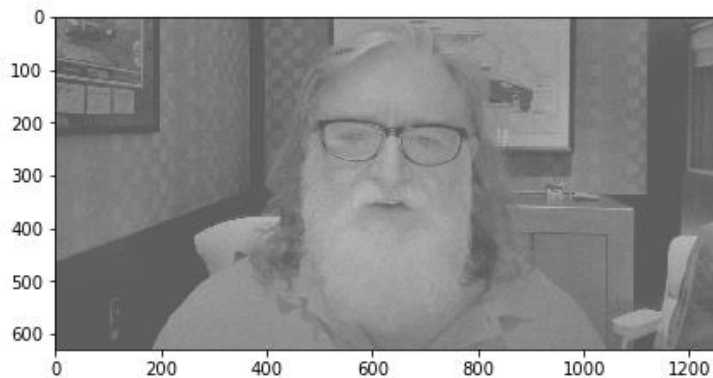


OpenCV содержит библиотеку общих функций искусственного интеллекта «Machine Learning Library» (MLL). Она служит, в основном, для распознавания фрагментов изображения и кластеризации.

**6. Определить динамический диапазон входного изображения.  
Осуществить линейное контрастирование входного изображения в  
заданный динамический диапазон яркостей**

```
In [22]: linear_contrast = imread("index.png")  
# linear_contrast  
imshow(linear_contrast)
```

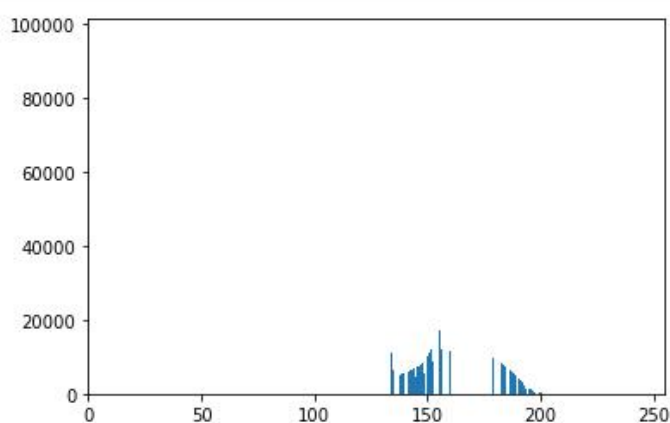
```
Out[22]: <matplotlib.image.AxesImage at 0x25e4b73fd30>
```



```
In [23]: linear_contrast.shape
```

```
Out[23]: (629, 1253, 4)
```

```
In [24]: # Расчет среднего значения из каналов RGB и сгладить до массива 1D  
vals = linear_contrast.mean(axis=2).flatten()  
# построение гистограммы с 255 интервалами  
b, bins, patches = plt.hist(vals, 255)  
plt.xlim([0, 255])  
plt.show()
```



Пусть минимальная и максимальная яркости исходного изображения равны  $x_{min}$  и  $x_{max}$  соответственно. При линейном контрастировании используется линейное поэлементное преобразование вида:



$$y = a * x + b$$

Параметры  $a$  и  $b$  определяются желаемыми значениями минимальной  $y_{min}$  и максимальной  $y_{max}$  выходной яркости.

$$\begin{cases} y_{min} = a * x_{min} + b \\ y_{max} = a * x_{max} + b \end{cases}$$

Тогда:

$$a = \frac{y_{max} - y_{min}}{x_{max} - x_{min}} ; b = y_{max} - a * x_{max}$$

Обычно в качестве рабочего используется диапазон от 0 до 255.

```
In [25]: f_min = linear_contrast.min()
         f_max = linear_contrast.max()
         print("f_min =", f_min)
         print("f_max =", f_max)
```

```
f_min = 93
f_max = 255
```

```
In [26]: # Приняв гистограмму исходного изображения, min и max значения интенсивности пикселей, делаем линейное контрастирование.
```

```
# Данная функция делает линейное контрастирование над input_image.
# Т.е. входной диапазон [f_min, f_max] растягивает на диапазон [g_min, g_max]

def linear_contrasting(input_image, f_min, f_max, g_min, g_max):
    a = (g_max - g_min) / (f_max - f_min)
    b = (g_min * f_max - g_max * f_min) / (f_max - f_min)
    output_image = (a * input_image + b).astype(int) # linear contrasting
    return output_image
```

```
In [27]: im_g = linear_contrasting(im, f_min, f_max, 0, 250)
```

```
In [28]: plt.figure()
         plt.title("Перед линейным контрастированием")
         plt.imshow(im)

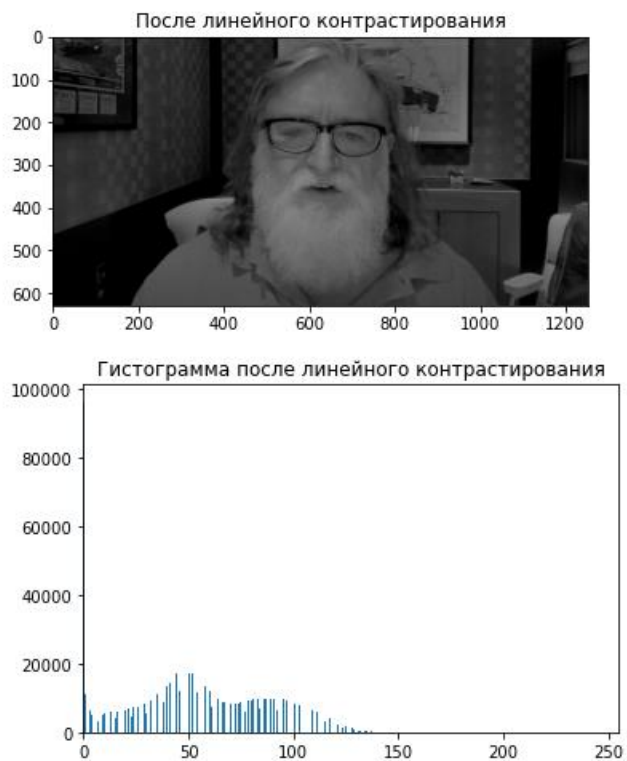
         plt.figure()
         plt.title("Гистограмма до линейного контрастирования")
         # Расчет среднего значения из каналов RGB и сгладить до массива 1D
         vals = im.mean(axis=2).flatten()
         # построение гистограммы с 255 интервалами
         plt.hist(vals, 255)
         plt.xlim([0, 255])
         plt.show()

         plt.figure()
         plt.title("После линейного контрастирования")
         plt.imshow(im_g)

         plt.figure()
         plt.title("Гистограмма после линейного контрастирования")
         # Расчет среднего значения из каналов RGB и сгладить до массива 1D
         vals = im_g.mean(axis=2).flatten()
         # построение гистограммы с 255 интервалами
         plt.hist(vals, 255)
         plt.xlim([0, 255])
         plt.show()
```



Как видно из гистограммы изображения, значения функции яркости расположены в узком диапазоне. По данной гистограмме непросто определить оптимальное значение для пороговой обработки.



Чтобы упростить задачу осуществили линейное контрастирование исходного изображения.

## 7. Сделать эквализацию гистограммы изображения

Для улучшения визуального качества к изображению надо применить такое преобразование, чтобы гистограмма результата содержала все возможные значения яркости и при этом в примерно одинаковом количестве.

```
In [29]: def cdf(im):
          values, bin_edges = np.histogram(im.ravel(), bins=range(257))
          new_value = values.cumsum()
          return new_value

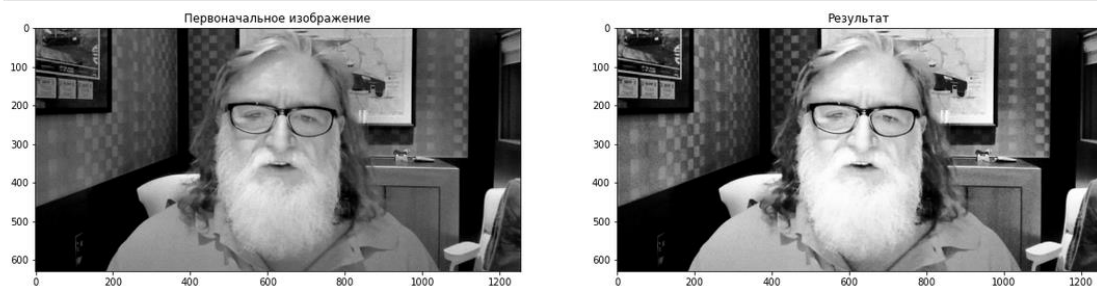
          def f(new_value, im):
              cdf_min = new_value[new_value != 0].min()
              col_px = new_value[-1] - 1
              return np.round((new_value[im] - cdf_min) * 255 / col_px)
```

```
In [30]: def equaliz(im):
          new_im = im.copy()
          new_value = cdf(new_im)
          img_eq = f(new_value, new_im)
          return np.int_(img_eq)
```

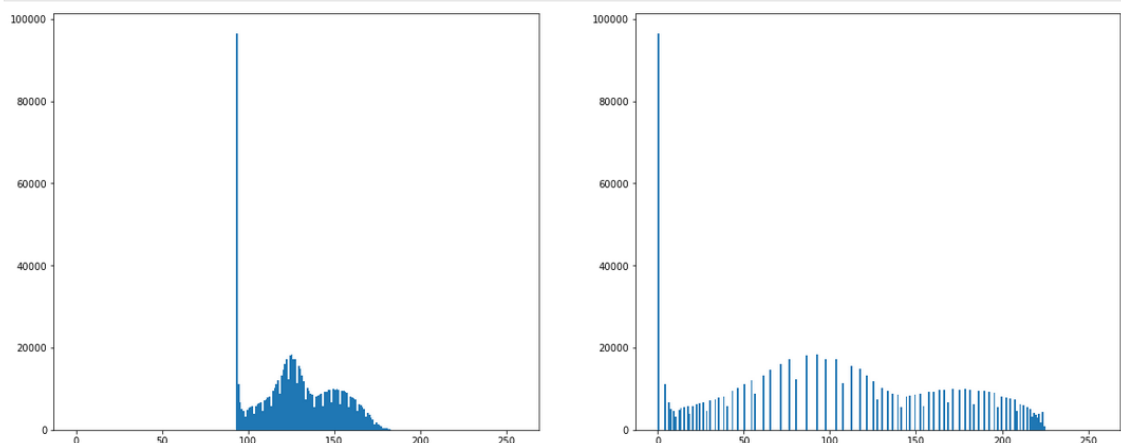
```
In [31]: imm = equaliz(img_gray)

          plt.figure(figsize=(20,20))
          plt.subplot(1, 2, 1)
          plt.title('Первоначальное изображение')
          plt.imshow(img_gray, cmap=plt.cm.gist_gray)

          plt.subplot(1, 2, 2)
          plt.title('Результат')
          plt.imshow(imm, cmap=plt.cm.gist_gray)
          plt.show()
```



```
In [32]: plt.figure(figsize=(20,8))
          plt.subplot(1, 2, 1)
          plt.hist(img_gray.ravel(), 256, [0, 256])
          plt.subplot(1, 2, 2)
          plt.hist(imm.ravel(), 256, [0, 256])
          plt.show()
```



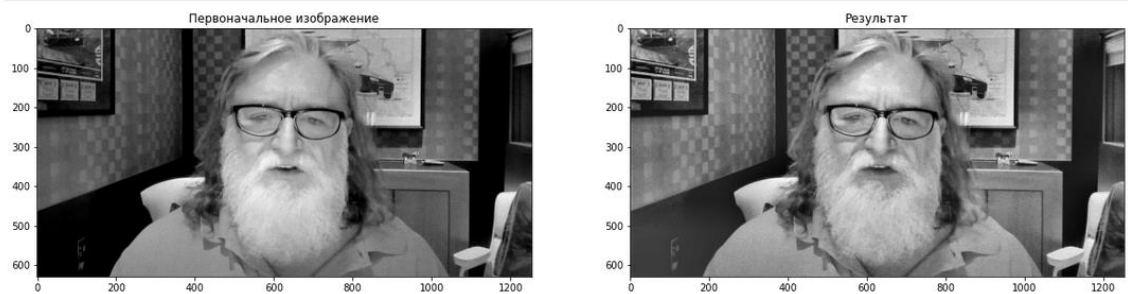
## 8. Сделать эквализацию методом CLAHE (Функция OpenCV)

Библиотека OpenCV также позволяет совершать эквализацию с помощью метода CLAHE - Contrast-limited Adaptive Histogram Equalization.

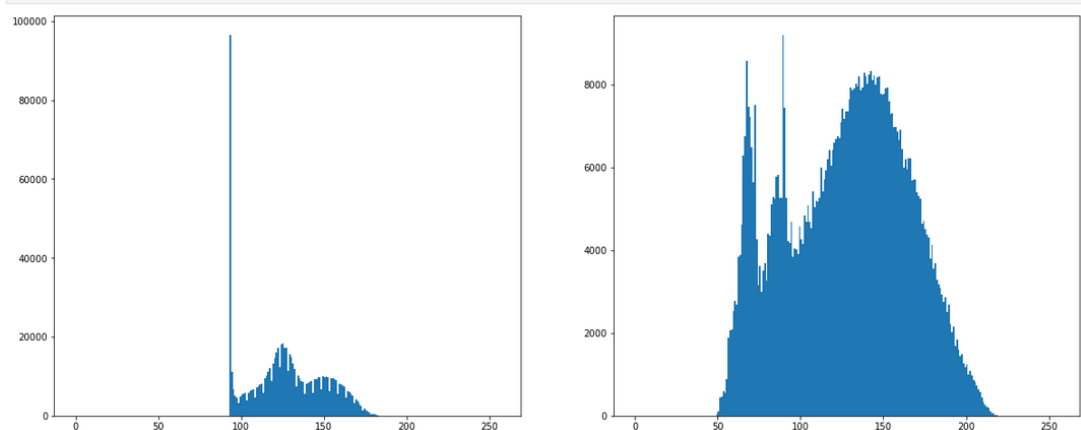
```
In [33]: clahe = cv2.createCLAHE(clipLimit = 2)
```

```
In [34]: imm = clahe.apply(img_gray)
plt.figure(figsize=(20,20))
plt.subplot(1, 2, 1)
plt.title('Первоначальное изображение')
plt.imshow(img_gray, cmap=plt.cm.gray)

plt.subplot(1, 2, 2)
plt.title('Результат')
plt.imshow(imm, cmap=plt.cm.gray)
plt.show()
```



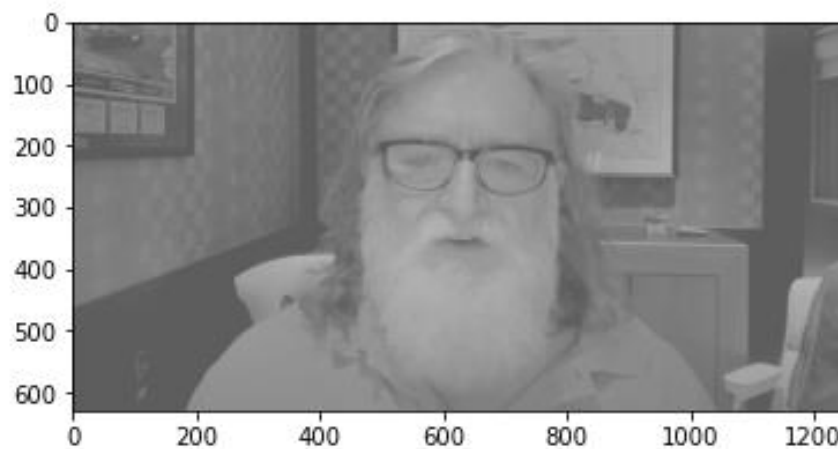
```
In [35]: plt.figure(figsize=(20,8))
plt.subplot(1, 2, 1)
plt.hist(img_gray.ravel(), 256, [0, 256])
plt.subplot(1, 2, 2)
plt.hist(imm.ravel(), 256, [0, 256])
plt.show()
```



## 9. Осуществить препарирование изображения с заданной препарирующей функцией

```
In [36]: # Берем исходное изображение из задания  
im = cv2.imread('index.png')  
plt.imshow(im)
```

```
Out[36]: <matplotlib.image.AxesImage at 0x25e4ac2afd0>
```



```
In [37]: # Размерность  
im.shape
```

```
Out[37]: (629, 1253, 3)
```

In [38]:

```
# проверяем работу функции на заданном массиве
```

```
arr = np.random.randint(0, 255, (6, 6))
print("arr =\n", arr)

im_prep_test = prepare_v3(arr, prepare_func_v3)

print("\nim_prep_test =\n", im_prep_test)
```

```
arr =
[[134  87  97   9 150 136]
 [137  32 162 130 114  24]
 [ 38 163 197   1  58  39]
 [185  66   3  54  67 167]
 [241  62 159 223 157  65]
 [162  12 144  13  46  26]]
```

```
im_prep_test =
[[ 91  11  28   0 119  95]
 [ 96   0 139  85  57   0]
 [  0 141 198   0   0   0]
 [178   0   0   0   0 147]
 [  0   0 134 243 130   0]
 [139   0 108   0   0   0]]
```

In [39]:

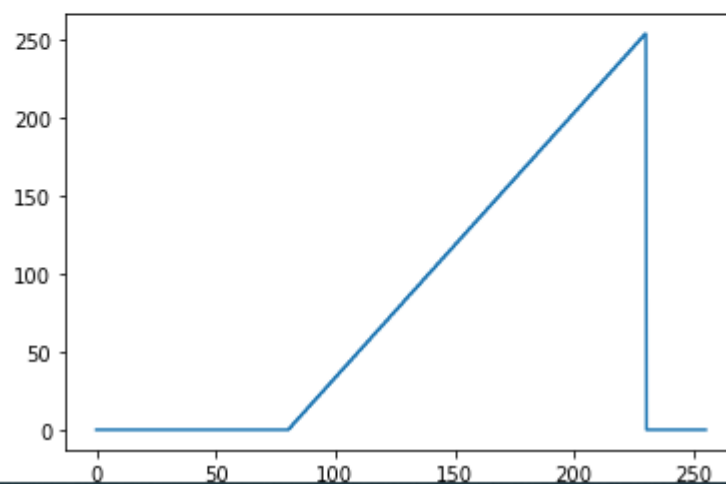
```
dot1 = 80
dot2 = 230

x = np.linspace(0, 255, 1000)

y = prepare_v3(x, prepare_func_v3)

plt.figure()
plt.plot(x, y)

im_prep = prepare_v3(im, prepare_func_v3)
```



In [40]:

```
plt.figure()
plt.title("До обработки")
plt.imshow(im)
plt.show()

plt.figure()
plt.title("Гистограмма до препарирования")

# Расчет среднего значения из каналов RGB и сгладить до массива 1D

vals = im.mean(axis=2).flatten()

# график гистограммы с 255 ячейками

plt.hist(vals, 255)
plt.xlim([0,255])
plt.show()

plt.figure()
plt.title("После обработки")
plt.imshow(im_prep)
plt.show()

plt.figure()
plt.title("Гистограмма после препарирования")

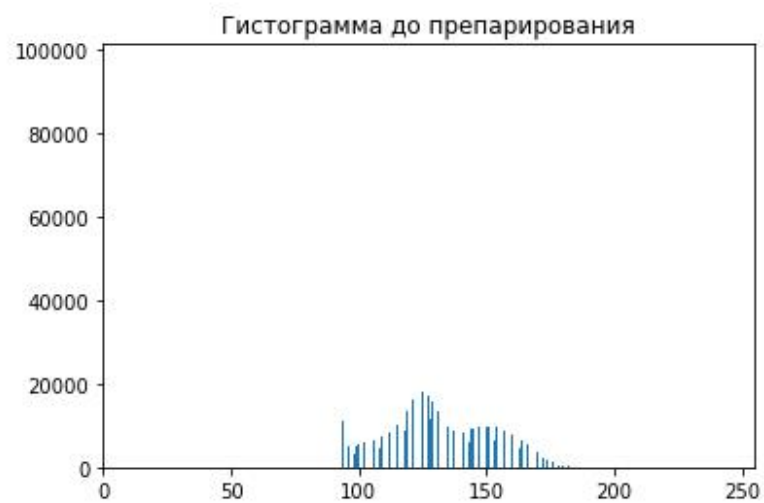
# Расчет среднего значения из каналов RGB и сгладить до массива 1D

vals = im_prep.mean(axis=2).flatten()

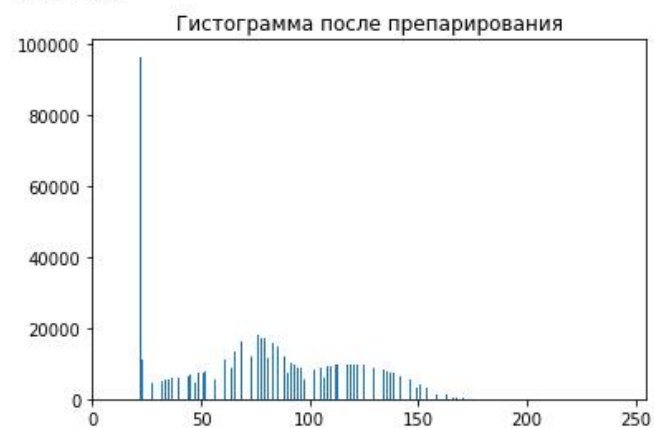
# график гистограммы с 255 ячейками

plt.hist(vals, 255)
plt.xlim([0,255])
plt.show

print("dot1 =", dot1)
print("dot2 =", dot2)
```



dot1 = 80  
dot2 = 230





## **ЗАКЛЮЧЕНИЕ**

В ходе лабораторной работы было изучено взаимодействие с изображениями в языке Python. Были совершены линейное контрастирование, пороговая обработка и поэлементное преобразование исходного изображения. Были построены и проанализированы гистограммы исходного изображения и преобразованных изображений.