Студент гр. РИМ-181226 Бабикова Евгения Витальевна

---

3. Коррекция яркости и цвета

Импорт необходимых библиотек и модулей

In [2]:

```
from skimage.io import imread, imsave, imshow
from skimage import img_as_ubyte, img_as_float
import numpy as np
from matplotlib.pyplot import hist
import matplotlib.pyplot as plt
```

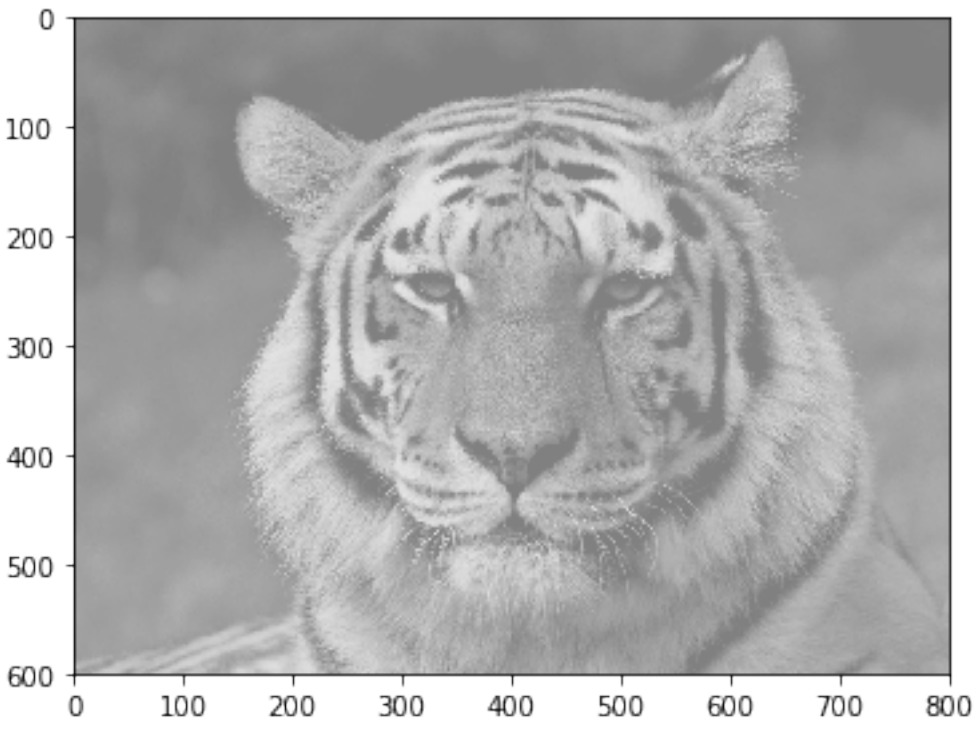
3.1. Линейная коррекция контраста

3.1.1. Автоконтраст черно-белого изображения

Считывание исходного малоконтрастного изображения.

In [4]:

```
img = imread('tiger-low-contrast.png')
imshow(img);
```



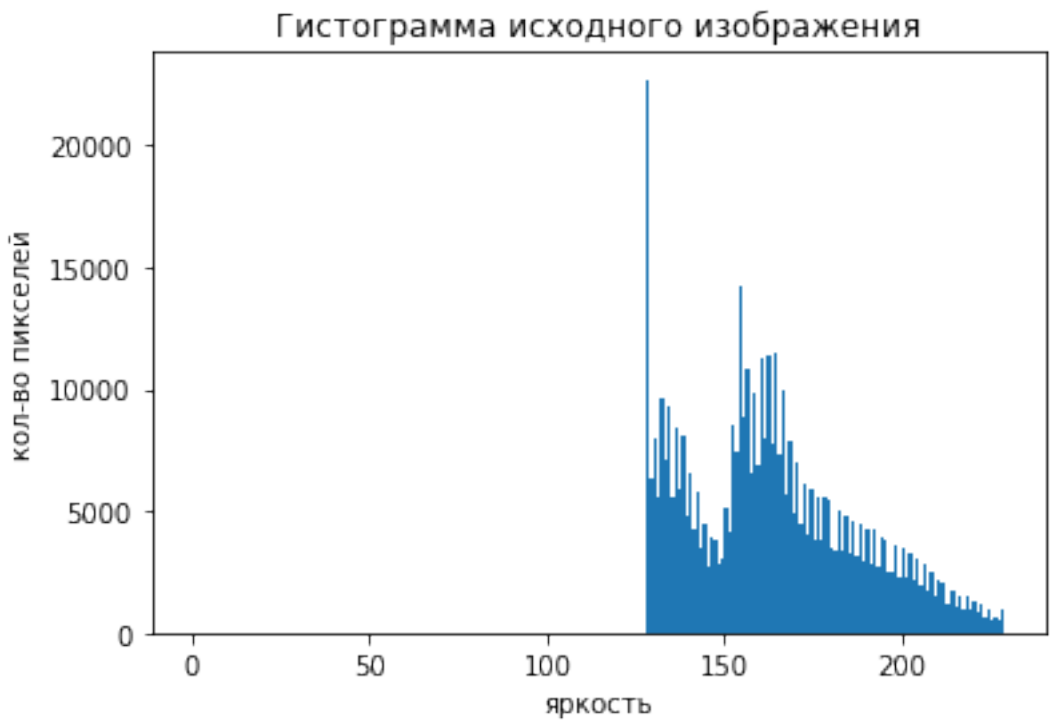
Построение гистограммы данного изображения.

Внешний вид гистограммы – расческа. Значению яркости 128 соответсвует большее кол-во пикселей, чем 129. Это происходит из-за того, что значения яркости разделили на 2 и в следсвтие округления, большее кол-во пикселей принимает 0 и четные значения яркости.

Видно, что пиксели данного изображения принимают значения из ограниченного диапазона: от 128 до 230. Это неконтратное изображение. Контрастность необходимо восстановить.

In [5]:

```
values, bin_edges, patches = hist(img.ravel(), bins=range(230))
plt.title('Гистограмма исходного изображения')
plt.xlabel('яркость')
plt.ylabel('кол-во пикселей');
```



Определение максимального и минимального значений пикселя в данном изображении. Для этого необходимо преобразовать входной массив – матрица размером 600*800 – в одномерный массив в 480 000 элементов.

In [6]:

```
x_min = min(img.ravel())
x_max = max(img.ravel())
```

Применение к каждому пикселю формулу линейного выравнивания яркости

$$f(x) = (x - x_{min}) \cdot \frac{255}{x_{max} - x_{min}}.$$

В первую очередь необходимо сместить значения пикселей в начало оси, для этого из каждого пикселя вычитается минимальное значение (значение, где гистограмма начинается). Затем нужно "растянуть" гистограмму на всю ось интенсивности пикселей, чтобы они принимали все возможные значения. Для этого необходим коэффициент.

In [7]:

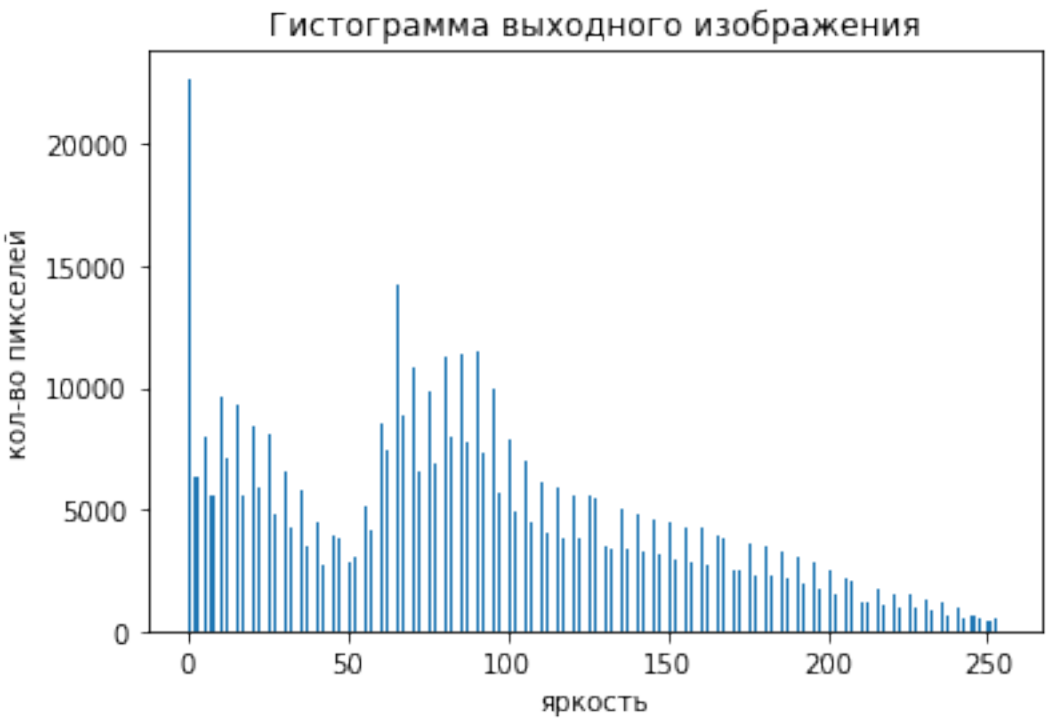
```
coef = 255/(x_max - x_min)
img_out = ((img - x_min) *coef).astype('uint8')
```

Переменная img_out содержит вещественные числа типа `float64` . Необходимо их преобразовать к типу `uint8` – целочисленные числа без знака в диапазоне от 0 до 255.

Построение гистограммы изображения, яркость которого восстановлена. По графику видно, что, во-первых, гистограмма сдвинута в начало координат и минимальным значением яркости является 0, а максимальным – 255. Во-вторых, яркость пикселей теперь растянута на всю ось значений – пиксели принимают все возможные значения яркости.

In [8]:

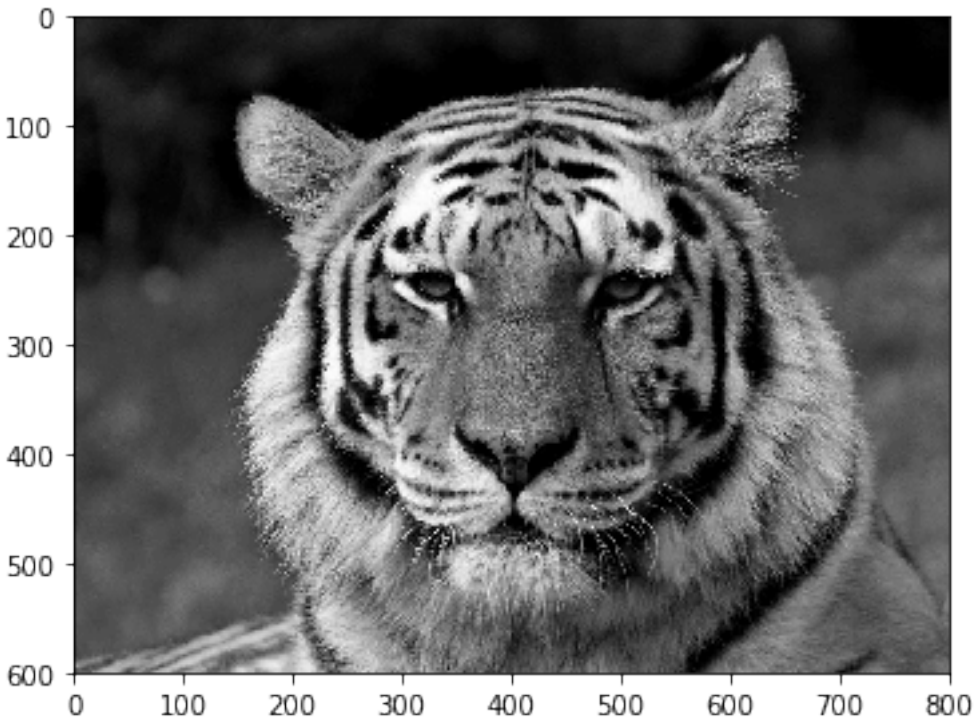
```
values_out, bin_edges_out, patches_out = hist(img_out.ravel(), bins=range(255)
)
plt.title('Гистограмма выходного изображения')
plt.xlabel('яркость')
plt.ylabel('кол-во пикселей');
```



Просмотр полученного изображения.

In [9]:

```
imshow(img_out);
```



Сравнение полученного изображения и образец.

In [10]:

```
img_sample = imread('tiger-high-contrast.png')
np.array_equal(img_out,img_sample)
```

Out[10]:

```
True
```

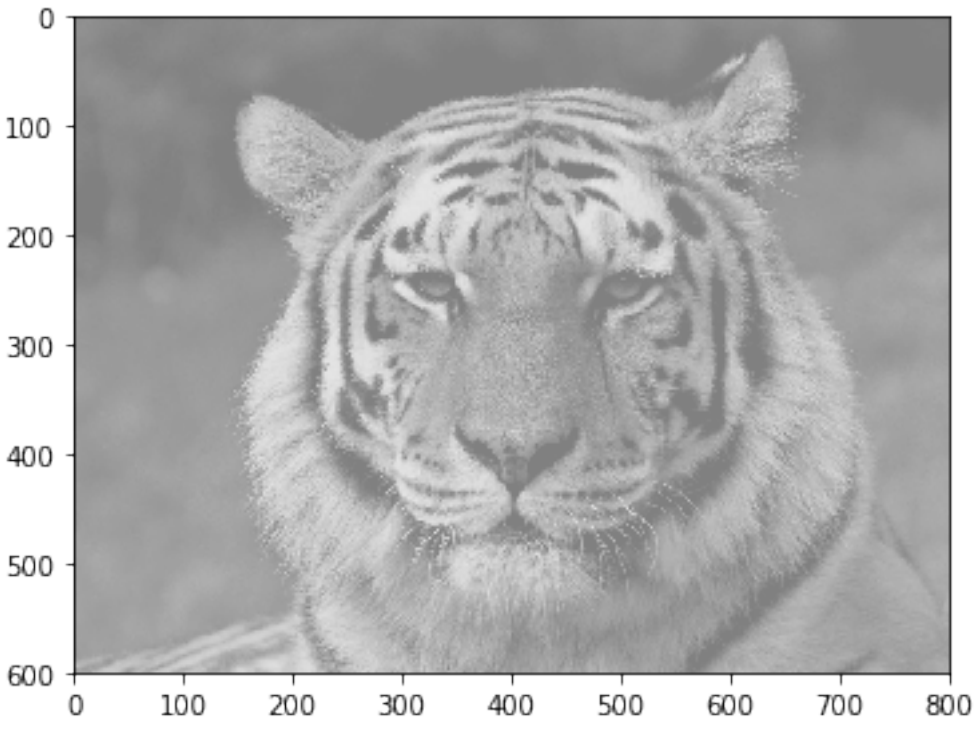
3.1.2. Устойчивый автоконтраст черно-белого изображения

Дотупистим в неконтрастном изображении есть некоторое кол-во белых и черных пикселей. Минимальное зн-ие яркости пикселя – 0, а максимальное – 255. Тогда применение к данному изображению ленейной коррекции контраста не даст результата. Для восстановления контрастности изображения необходимо использовать другой подход.

Перед определением минимальной и максимальной яркости нужно "выкинуть" 5% самых темных и самых светлых пикселей. Для этого общее кол-во пикселей домножим на 5%. Результат – кол-во пикселей, которые необходимо отбросить. Он может получиться нецелым, а кол-во пикселей должно быть целым числом, поэтому округлим его.

In [12]:

```
img = imread('tiger-low-contrast.png')
imshow(img);
img = img.astype('float')
k = round(img.shape[0] * img.shape[1] * 0.05)
```



Для определения минимальной и максимальной яркости пикселей, данные значения необходимо отсортировать. Исходный многомерный массив представлен в виде вектора и отсортирован. Копирование необходимо для сохранение исходного массива (функция `sort()` перепишет его).

In [13]:

```
img_sort = img.copy()
img_sort = img_sort.ravel()
img_sort.sort()
```

Поиск минимальной и максимальной яркости необходимо начать с k-го элемента, следующего после тех самых "выкидываемых" 5% значений пикселей с начала массива, и проводить его до 5% "выкидываемых значений пикселей с конца массива."

In [14]:

```
x_min = img_sort[k]
x_max = img_sort[len(img_sort) - k]
print(x_min,x_max)
```

```
129.0 208.0
```

Применение формулы линейного выравнивания контрастности.

In [15]:

```
coef = 255/(x_max - x_min)
img_out = (img - x_min) * coef
```

Полученные значения могут быть меньше нуля и больше 255. Поэтому необходимо применить функцию `clip()` , чтобы ограничить яркости пикселей диапазоном [0,255].

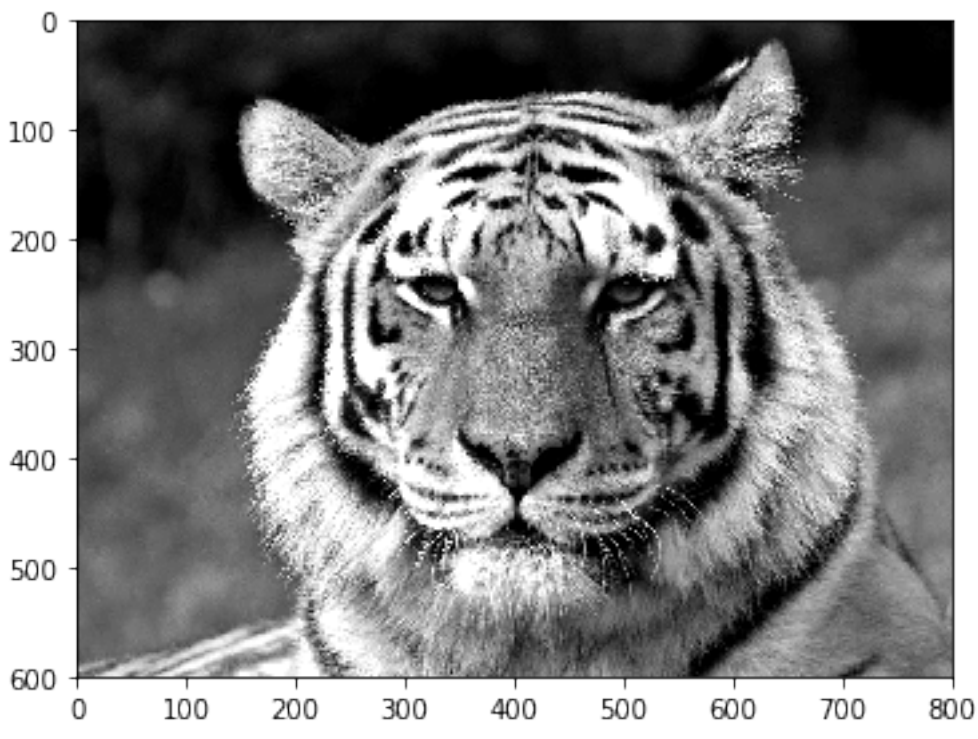
In [16]:

```
img_out = np.clip(img_out,0,255).astype('uint8')
```

Просмотр полученного изображения.

In [17]:

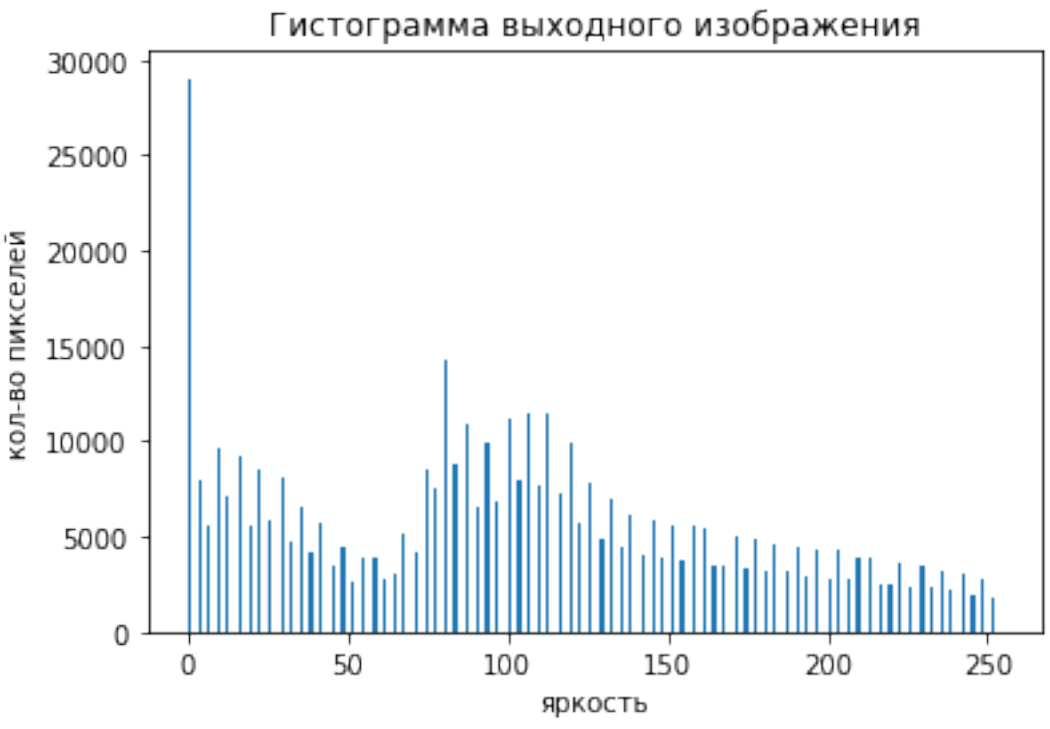
```
imshow(img_out);
```



Гистограмма выходного изображения.

In [18]:

```
values_out, bin_edges_out, patches_out = hist(img_out.ravel(), bins=range(255)
)
plt.title('Гистограмма выходного изображения')
plt.xlabel('яркость')
plt.ylabel('кол-во пикселей');
```



Сравнение полученного изображения и образца.

In [19]:

```
img_sample = imread('tiger-stable-contrast.png')
np.array_equal(img_out,img_sample)
```

Out[19]:

```
True
```

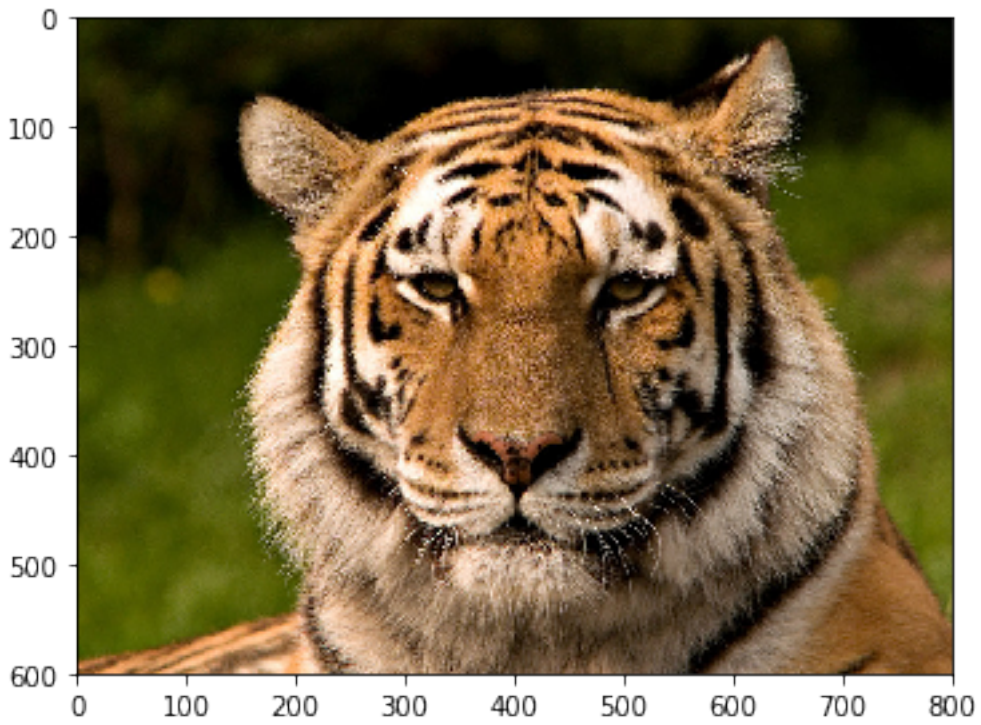
3.2. Коррекция контраста цветного изображения

3.2.1 Устойчивый цветной автоконтраст

Считывание цветного изображения.

In [20]:

```
img = imread('tiger-color.png')
imshow(img);
```



Перевод массива исходных значений пикселей в числа с плавающей запятой.

In [21]:

```
img = img_as_float(img)
```

Цветное изображение состоит из трех каналов RGB. Каждая компонента вносит свой вклад в яркость изображения. Для упрощения работы с цветными изображениями, необходимо перейти в пространство YUV, где Y – канал, отвечающий за яркость изображения, U - синяя цветовая компонента, V - красная цветовая компонента.

Реализация перехода из RGB в YUV.

In [22]:

```
Y = 0.2126 * img[:,:,0] + 0.7152 * img[:,:,1] + 0.0722 * img[:,:,2]
U = -0.0999 * img[:,:,0] - 0.336 * img[:,:,1] + 0.436 * img[:,:,2]
V = 0.615 * img[:,:,0] - 0.5586 * img[:,:,1] - 0.0563 * img[:,:,2]
```

Далее необходимо применить устойчивый автоконтраст изображения для канала Y.

In [23]:

```
k = round(int(Y.shape[0] * Y.shape[1] * 0.05))
Y_sort = Y.copy()
Y_sort = Y_sort.ravel()
Y_sort.sort()
Y_min = Y_sort[k]
Y_max = Y_sort[len(Y_sort) - k]
coef = 1/(Y_max - Y_min)
Y_out = (Y - Y_min) * coef
```

Обрезка значений канала Y от 0 до 1.

In [24]:

```
Y_out = np.clip(Y_out,0,1)
```

Перевод изображения из пространства YUV в RGB.

In [25]:

```
R = Y_out + 1.2803 * V
G = Y_out - 0.2148 * U - 0.3805 * V
B = Y_out + 2.1279 * U
```

Объединение полученных значений каналов RGB в один массив.

In [26]:

```
img_out = np.dstack((R,G,B))
```

При конвертации изображения в разные пространства могут появится числа, не помещающиеся в диапазон [0,1], поэтому полченные значения необходимо обрезать до нужного диапазона. Далее массив приводится к типу `ubyte` с диапазоном значений [0,255].

In [27]:

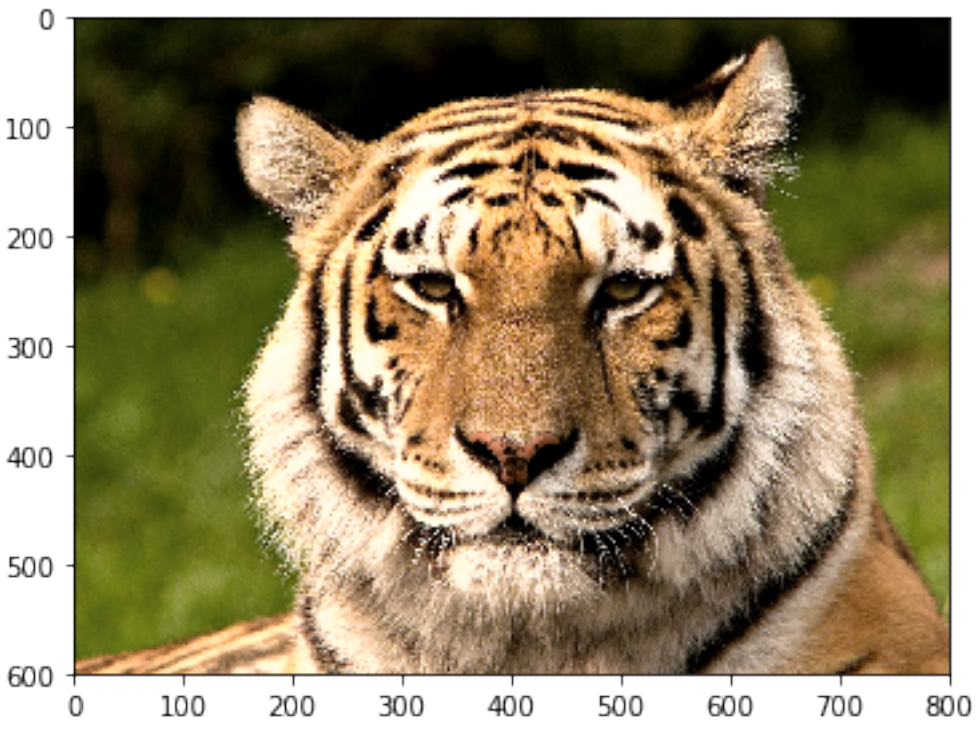
```
img_out = np.clip(img_out,0,1)
img_out = img_as_ubyte(img_out)
```

```
/anaconda3/lib/python3.6/site-packages/skimage/util/dtype.py:141:
UserWarning: Possible precision loss when converting from float64
to uint8
  .format(dtypeobj_in, dtypeobj_out))
```

Просмтор полученного изображения.

In [28]:

```
imshow(img_out);
```



Сравнение полученного изображения и образца.

In [29]:

```
img_sample = imread('tiger-color-stable-contrast.png')
np.array_equal(img_out,img_sample)
```

Out[29]:

```
True
```

3.3. Баланс белого. Модель «серого мира»

3.3.1 Преобразование серого мира

Считывание исходного изображения с нарушенным балансом белого.

In [31]:

```
img = imread('railroad.png')
imshow(img);
```

Перевод массива исходных значений пикселей в числа с плавающей запятой.

In [32]:

```
img = img_as_float(img)
```

Расчет значений для преобразования:

1. Средних значений каналов $\bar{R}$, $\bar{G}$, $\bar{B}$.
2. Среднего значения яркости изображения Avg.
3. Коэффициентов r_w, g_w, b_w.

In [33]:

```
R_avg = np.mean(img[:,:,0])
G_avg = np.mean(img[:,:,1])
B_avg = np.mean(img[:,:,2])
Avg = (R_avg + G_avg + B_avg)/3
r_w = R_avg/Avg
g_w = G_avg/Avg
b_w = B_avg/Avg
```

Деление каналов изображения на коэффициенты.

In [34]:

```
R = img[:,:,0]/r_w
G = img[:,:,1]/g_w
B = img[:,:,2]/b_w
```

Объединение полученных значений каналов в один массив.

In [35]:

```
img_out = np.dstack((R,G,B))
```

Обрезка полученных значений пикселей и преобразование массива в тип `ubyte` .

In [36]:

```
img_out = np.clip(img_out,0,1)
img_out = img_as_ubyte(img_out)
```

Просмотр изображения с восстановленным балансом белого.

In [37]:

```
imshow(img_out);
```

Сравнение полученного изображения и образца.

In [38]:

```
img_sample = imread('railroad-gray-world.png')
np.array_equal(img_out,img_sample)
```

Out[38]:

```
True
```

3.3. Выравнивание гистограммы

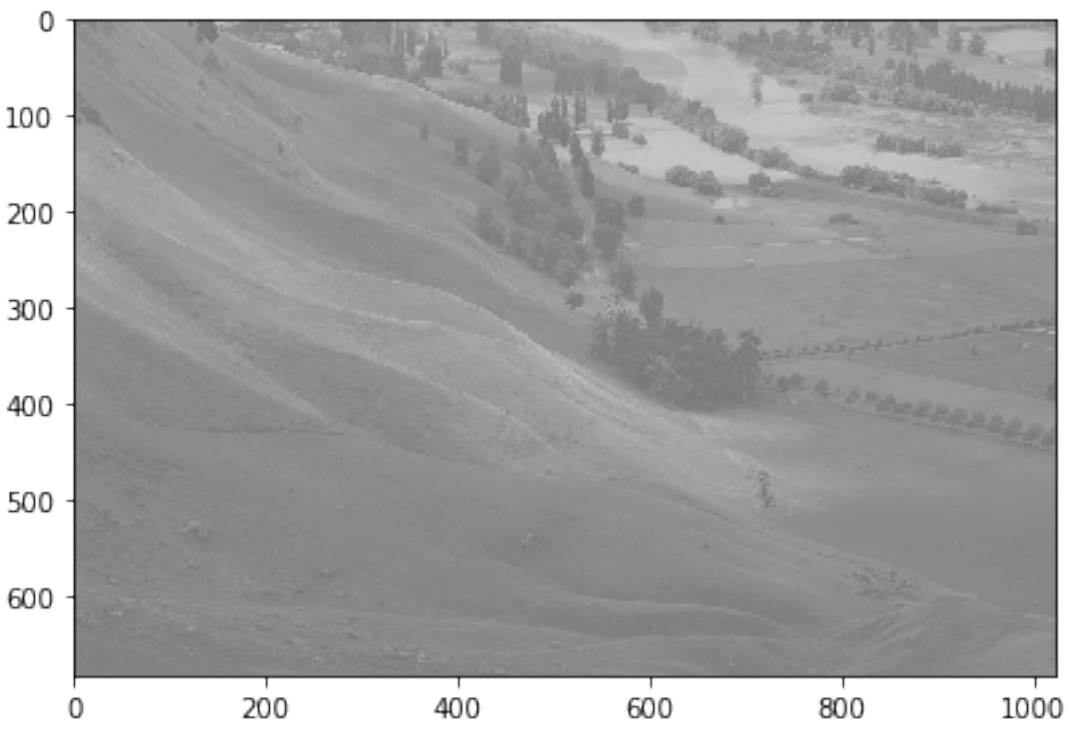
3.3.1 Выравнивание гистограммы

Выравниваение гистограммы – продвинутый метод повышения контрастности изображения.

Считывание исходного некотрастного изображения.

In [39]:

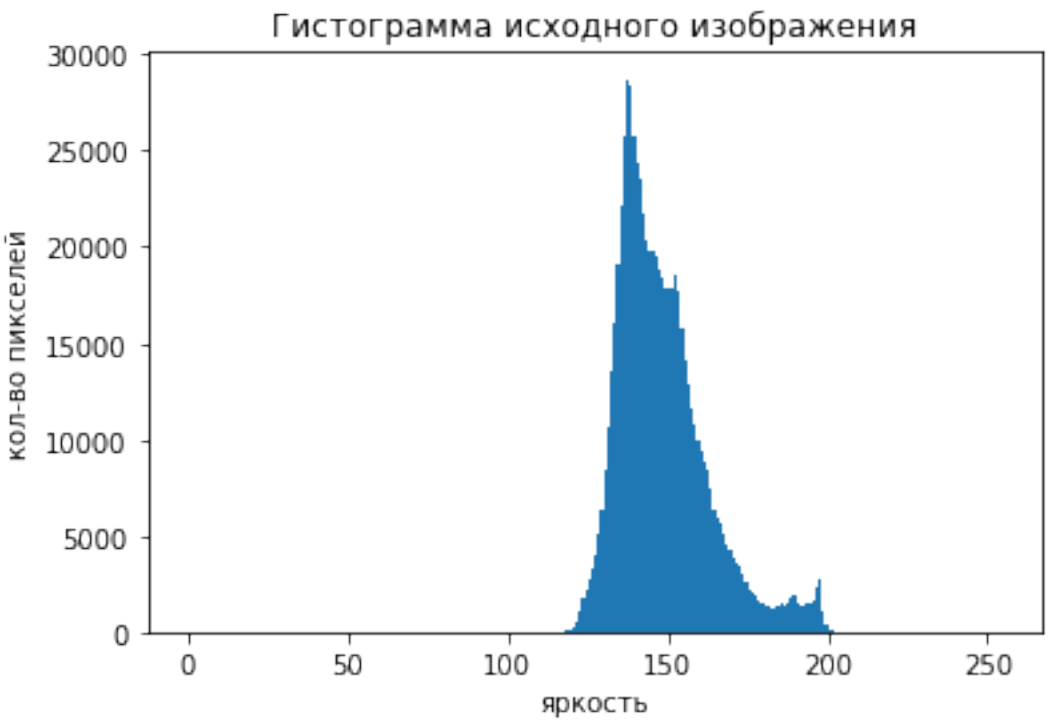
```
img = imread('landscape.png')
imshow(img);
```



Построение гисторграммы h исходного изображения.

In [40]:

```
values, bin_edges, patches = hist(img.ravel(), bins=range(255))
plt.title('Гистограмма исходного изображения')
plt.xlabel('яркость')
plt.ylabel('кол-во пикселей');
```



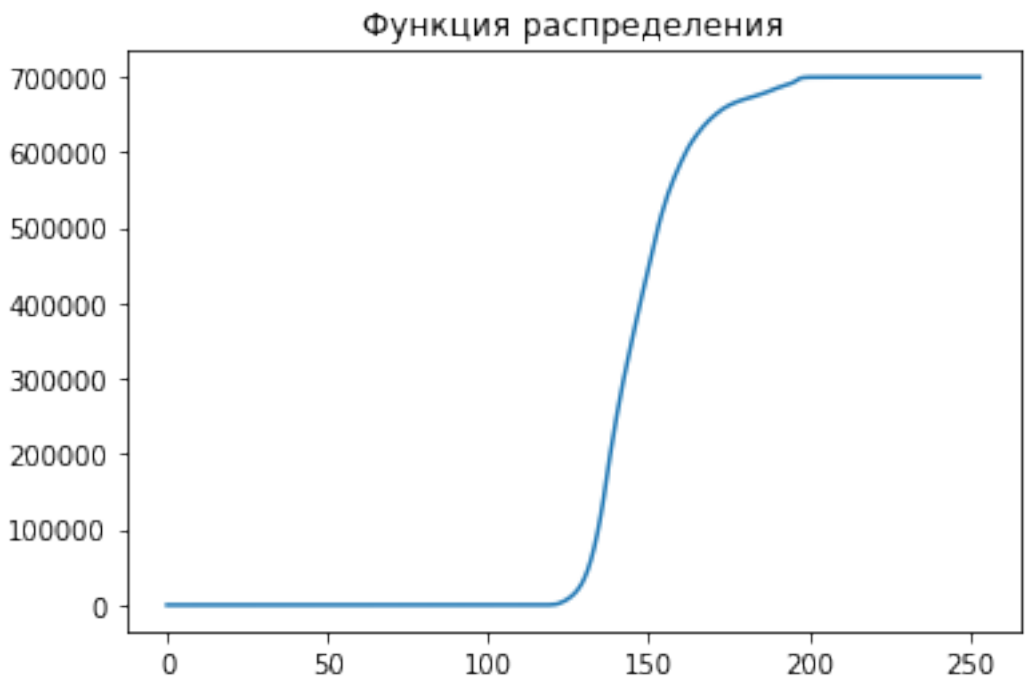
Из гистограммы видно, что пиксели данного изображения принимают значения из ограниченного диапазона: от 128 до 200. Это неконтратное изображение. Контрастность необходимо восстановить.

Функция распределения определяется как сумма ячеек гистограммы:

$$cdf(x) = h(0) + h(1) + \ldots + h(x).$$

In [41]:

```
cdf = np.cumsum(values)
plt.title('Функция распределения')
plt.plot(cdf);
```



Функция распределения возрастает резко и неравномерно, что соответствует малоконтрастному изображению.

Реализация формулы для повышения контрастности:

$$f(x) = round(\frac{cdf(x) - cdf_{min}}{pix - 1} \cdot 255),$$

где pix – кол-во пикселей в изображении.

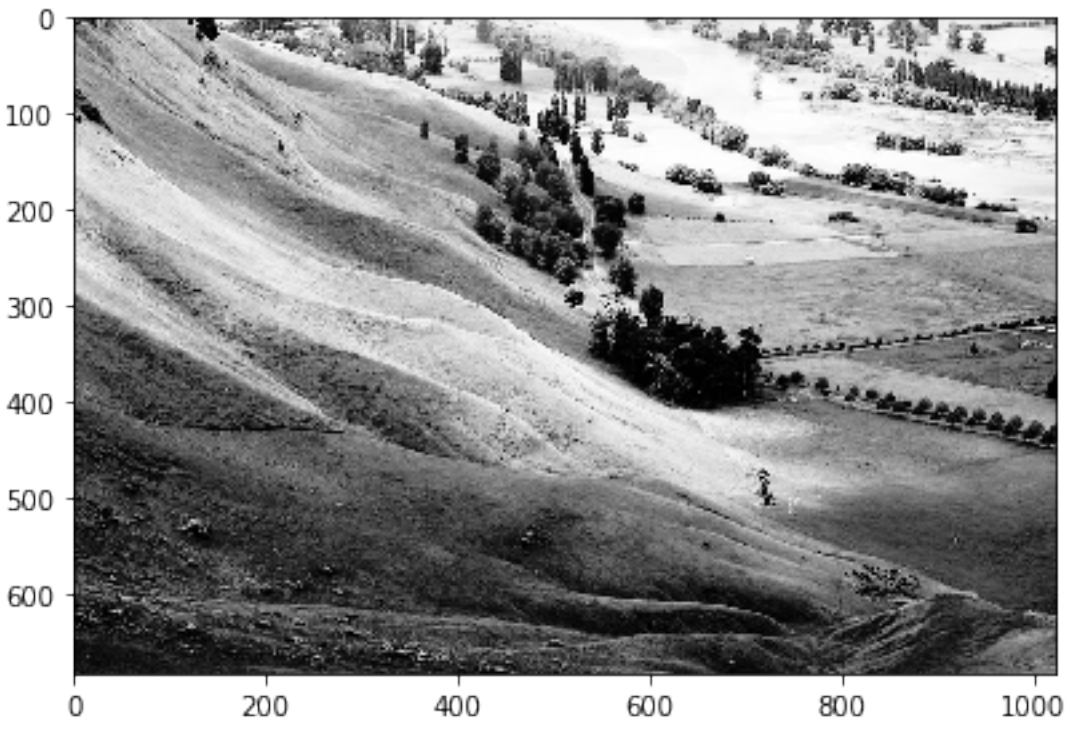
In [42]:

```
img_out = img.copy()
min_cdf = np.min(cdf[np.nonzero(cdf)]) # определяем минимальное значение cdf н
еравное 0
coef = 255/(img.size - 1)
for i in range(img_out.shape[0]):
    for j in range(img_out.shape[1]):
        img_1 = cdf[img_out[i,j]] - min_cdf
        img_1 = img_1 * coef
        img_out[i,j] = img_1.round()
```

Преобразование полученных значений в тип `ubyte` и просмотр изображения с выравненной гистограммой.

In [43]:

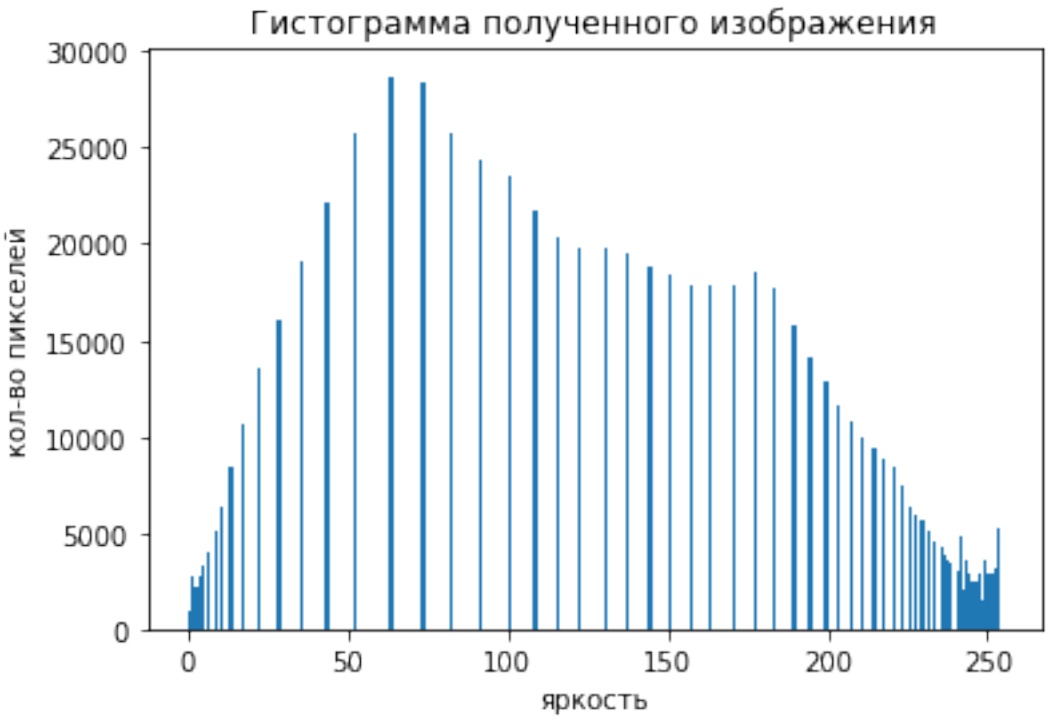
```
img_out = img_as_ubyte(img_out)
imshow(img_out);
```



Построение гистограммы и функции распределения полученного изображения.

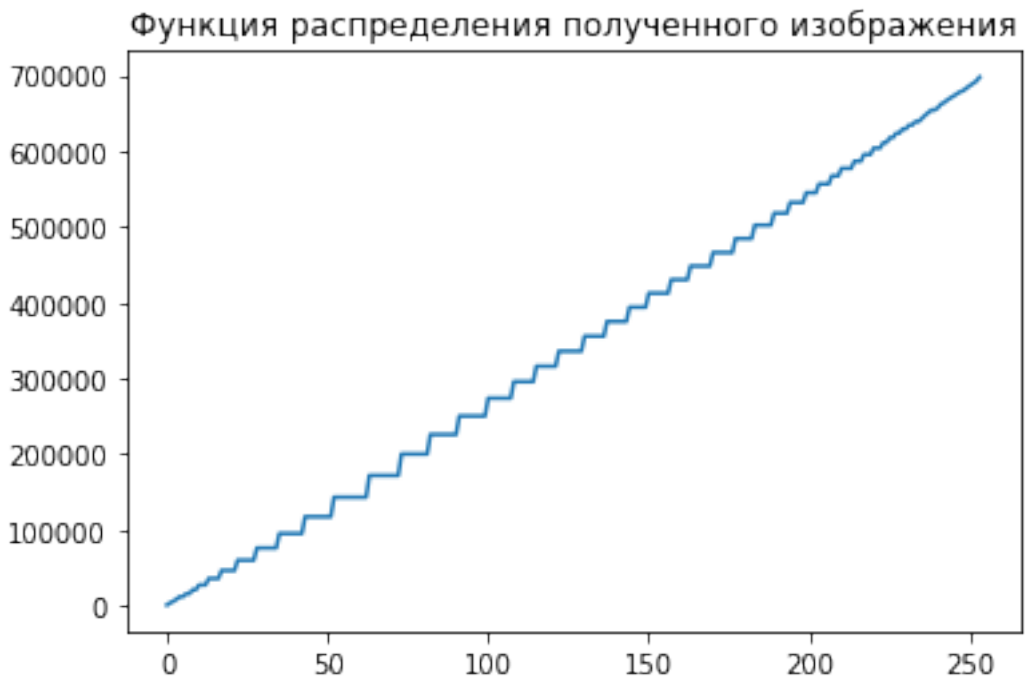
In [44]:

```
values_out, bin_edges_out, patches_out = hist(img_out.ravel(), bins=range(255)
)
plt.title('Гистограмма полученного изображения')
plt.xlabel('яркость')
plt.ylabel('кол-во пикселей');
```



In [45]:

```
cdf_out = np.cumsum(values_out)
plt.title('Функция распределения полученного изображения')
plt.plot(cdf_out);
```



Функция распределения имеет вид близкий к линейной функции.

Сравнение полученного изображения и образца.

In [46]:

```
img_sample = imread('landscape-histeq.png')
np.array_equal(img_out, img_sample)
```

Out[46]:

```
True
```