



НИУ ИТМО

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

“Гистограммы, профили и проекции”

Выполнили:

Александр Иванов, R3238

Никита Братушка, R3238

Ани Аракелян, R3242

Преподаватель:

Шаветов С. В.

Санкт-Петербург, 2024

Содержание

1 Используемые функции	2
1.1 Функция для вычисления гистограммы изображения	2
1.2 Функция для отображения изображения и его гистограмм	2
2 Преобразования изображений	4
2.1 Исходное изображение	4
2.2 Линейное выравнивание гистограммы	5
2.3 Арифметические операции над изображениями	6
2.4 Растворение динамического диапазона	7
2.5 Равномерное преобразование	8
2.6 Экспоненциальное преобразование	10
2.7 Преобразование по закону Рэлея	11
2.8 Преобразование по закону степени 2/3	13
2.9 Гиперболическое преобразование	14
2.10 Таблица поиска	15
3 Профили изображения	17
3.1 Профиль изображения вдоль горизонтальной линии	17
4 Проекция изображения	18
4.1 Проекция изображения на оси	18
5 Вывод	22

1 Используемые функции

1.1 Функция для вычисления гистограммы изображения

Для вычисления гистограммы изображений была написана следующая функция:

```
def calc_hist_normalised(img):
    histSize = 256
    histRange = (0, 256)
    # calculate the histograms
    b_hist = cv2.calcHist([img], [0], None, [histSize], histRange) /
        (img.shape[0] * img.shape[1])
    g_hist = cv2.calcHist([img], [1], None, [histSize], histRange) /
        (img.shape[0] * img.shape[1])
    r_hist = cv2.calcHist([img], [2], None, [histSize], histRange) /
        (img.shape[0] * img.shape[1])

    return b_hist, g_hist, r_hist
```

В качестве аргумента она принимает изображение, для которого необходимо вычислить гистограмму. Возвращает функция нормализованные гистограммы для каждого канала изображения. Под нормализованной гистограммой понимается гистограмма, в которой каждое значение поделено на общее количество пикселей в изображении.

1.2 Функция для отображения изображения и его гистограмм

Для отображения изображения, его гистограмм и результата преобразования использовалась следующая функция:

```
# add histogram and image to the same plot
def show_image_with_hist(img, title="Image"):
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # normal colors to
    display
    b_hist, g_hist, r_hist = calc_hist_normalised(img) # calculate the
    histograms
```

```

cumulative_hist_b, cumulative_hist_g, cumulative_hist_r =
    np.cumsum(b_hist), np.cumsum(g_hist), np.cumsum(r_hist) # calculate
the cumulative histograms

gs = plt.GridSpec(2, 4, width_ratios=[3, 1, 1, 1])

plt.figure(figsize=(13, 5))

plt.suptitle(title, fontsize=16)
ax0 = plt.subplot(gs[:, 0]) # for image
ax0.set_title('Image')
ax1 = plt.subplot(gs[0, 1:4]) # for rgb hist
ax1.set_title('RGB Histogram')

ax2 = plt.subplot(gs[1, 1:3])
ax2.set_title('Cumulative RGB Histogram')
ax3 = plt.subplot(gs[1, 3])
ax3.set_title('Average Histogram')

# set the x axis limits for the histogram subplots and grid
for ax in [ax1, ax2, ax3]:
    ax.set_xlim([0, 256])
    ax.grid(True)

# display the image
ax0.imshow(img_rgb)
ax0.axis('off')

# all 3 histograms
ax1.plot(b_hist, color='b')
ax1.plot(g_hist, color='g')
ax1.plot(r_hist, color='r')

# cumulative histograms
ax2.plot(cumulative_hist_b, color='b')
ax2.plot(cumulative_hist_g, color='g')
ax2.plot(cumulative_hist_r, color='r')

```

```

# add 3 colors hist
avg_hits = (b_hist + g_hist + r_hist) / 3
ax3.hist(np.arange(0, 256), bins=256, weights=avg_hits, color='black')

# adjust the layout
plt.subplots_adjust(wspace=0.3, hspace=0.3)
plt.subplots_adjust(left=0.05, right=0.95)

```

Во всех следующих функциях для преобразования изображений оно сначала раскладывается на каналы, затем преобразование применяется к каждому каналу, после чего каналы собираются обратно в изображение. В некоторых случаях это позволяет применить различные значения преобразования к различным каналам изображения, что позволяет получить более интересные результаты.

2 Преобразования изображений

2.1 Исходное изображение

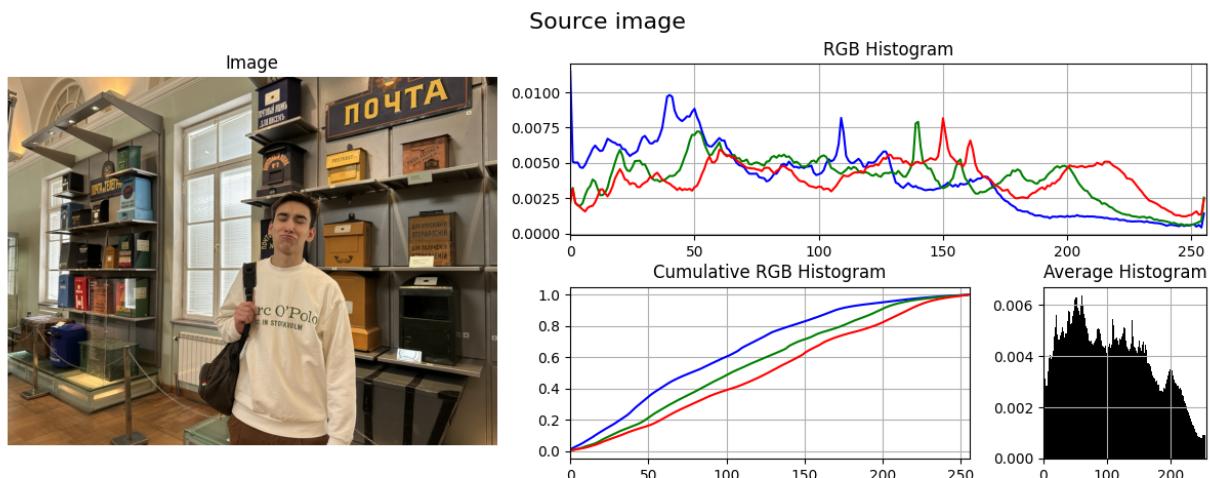


Рис. 1: Исходное изображение

На рисунке 1 представлено исходное изображение, для которого необходимо вычислить гистограмму.

Кроме того, в правой части рисунка 1 представлены гистограммы для каждого канала изображения, коммулятивная гистограмма и средняя гистограмма для всех каналов.

2.2 Линейное выравнивание гистограммы

Суть линейного выравнивания гистограммы заключается в использовании коммулятивной гистограммы для вычисления нового значения пикселя.

Исходный код функции для линейного выравнивания гистограммы:

```
def linear_leveling_transformation(img):  
    # split the image into its 3 channels  
    b, g, r = cv2.split(img)  
  
    # calculate the histograms  
    hist = calc_hist_normalised(img)  
  
    # calculate the cumulative histograms  
    cumulative_histogram_b = np.cumsum(hist[0])  
    cumulative_histogram_g = np.cumsum(hist[1])  
    cumulative_histogram_r = np.cumsum(hist[2])  
  
    # apply the transformation  
    b = np.clip(255 * cumulative_histogram_b[b], 0, 255)  
    g = np.clip(255 * cumulative_histogram_g[g], 0, 255)  
    r = np.clip(255 * cumulative_histogram_r[r], 0, 255)  
  
    # merge the channels back  
    return cv2.merge([b, g, r]).astype(np.uint8)
```

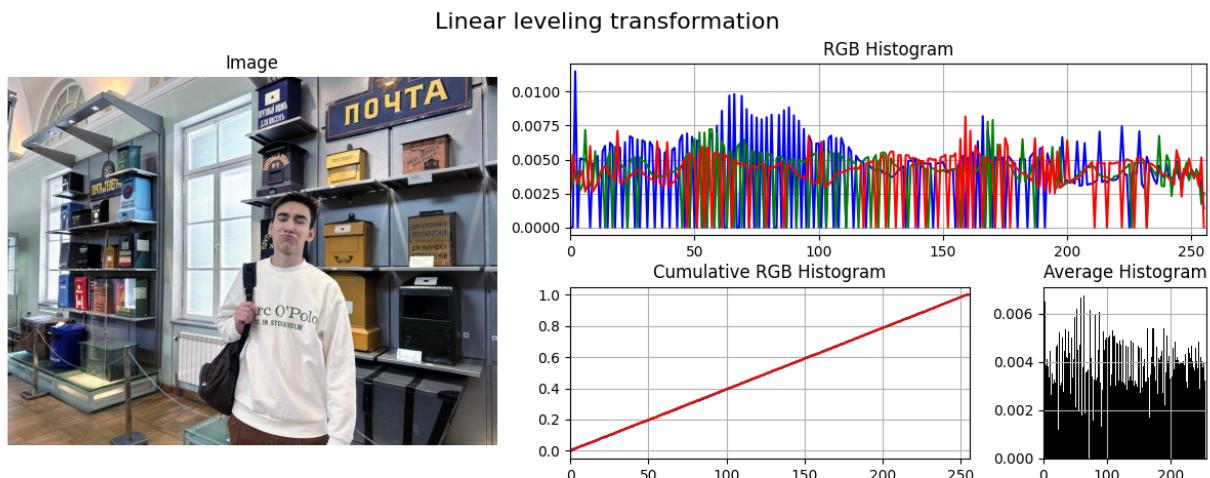


Рис. 2: Результат линейного выравнивания гистограммы

На рисунке 2 представлен результат применения линейного выравнивания гистограммы к исходному изображению. Как видно, коммулятивная гистограмма стала представлять из себя линейную функцию, что и является результатом применения линейного выравнивания гистограммы.

Цвета изображения при этом стали более холодными, сама гистограмма стала прерывистой, что связано с тем, что на преобразованном изображении не могут появиться некоторые значения пикселей.

2.3 Арифметические операции над изображениями

Для увеличения детализации некоторых областей изображения, можно использовать арифметические операции над изображениями. Например, для увеличения детализации теней можно использовать смещение цветов изображения в сторону светлых тонов.

Исходный код функции для арифметических операций над изображениями:

```
def aritmetic_transformation(img, b_delta, g_delta, r_delta):
    # split the image into its 3 channels
    b, g, r = cv2.split(img)

    # add the delta to each channel and divide by 255
    b = (b + b_delta)
    g = (g + g_delta)
    r = (r + r_delta)

    # merge the channels back
    return cv2.merge([b, g, r])
```

Так как мы используем цветное изображение, то для каждого канала можно использовать свой коэффициент смещения.

На рисунке 3 представлен результат применения арифметических операций над изображением. К красному, зеленому и синему каналам изображения были добавлены значения 30, 20 и 10 соответственно.

Как видно, изображение стало более светлым, что связано с тем, что мы добавили ко всем

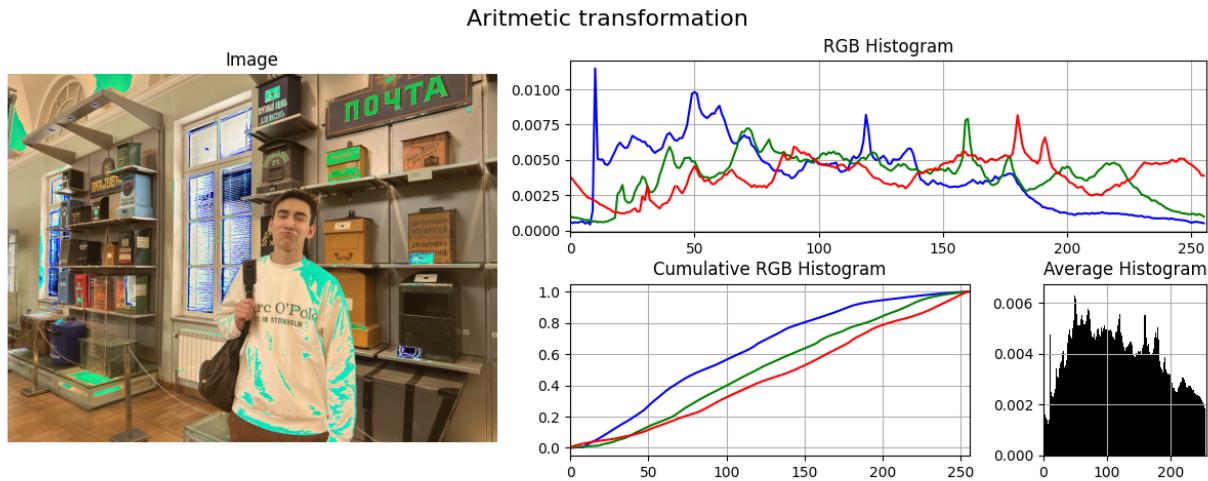


Рис. 3: Результат арифметических операций над изображением

каналам изображения некоторое значение. Так же видны зелено-синие пятна на изображении, что связано с тем, что мы добавили ко всем каналам значения, что привело к изменению цветового баланса изображения.

На гистограмме так же видно, что на изображении стало больше светлых пикселей.

2.4 Растворение динамического диапазона

Данное преобразование выполняется согласно следующему закону:

$$I_{new} = \left(\frac{I - I_{min}}{I_{max} - I_{min}} \right)^\alpha \quad (1)$$

где I – исходное изображение, I_{new} – преобразованное изображение, I_{min} и I_{max} – минимальное и максимальное значение пикселя на изображении, α – коэффициент нелинейности.

Исходный код функции для растворения динамического диапазона:

```
def dynamic_range_expansion_transformation(img, aplha):
    # find maximum and minimum values for each channel

    # if the image is of type uint8, convert it to float64
    if img.dtype == 'uint8':
        img_converted = img.astype(np.float64) / 255
```

```

b, g, r = cv2.split(img_converted)

b_min, b_max = b.min(), b.max()
g_min, g_max = g.min(), g.max()
r_min, r_max = r.min(), r.max()

# apply the transformation

b = np.clip(((b - b_min) / (b_max - b_min)) ** aplha, 0, 1)
g = np.clip(((g - g_min) / (g_max - g_min)) ** aplha, 0, 1)
r = np.clip(((r - r_min) / (r_max - r_min)) ** aplha, 0, 1)

img_transformed = cv2.merge([b, g, r]) # merge the channels back

# convert the image back to uint8 if it was initially of that type
if img.dtype == 'uint8':
    img_transformed = (255 * img_transformed).clip(0,
255).astype(np.uint8)

return img_transformed

```

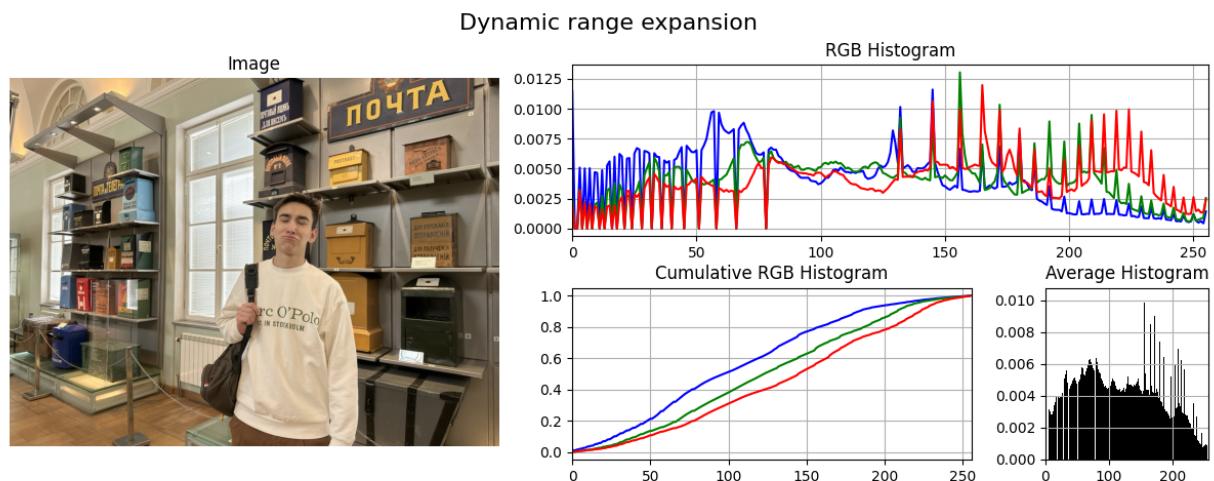


Рис. 4: Результат растяжения динамического диапазона

На рисунке 4 представлен результат применения растяжения динамического диапазона к исходному изображению. Как видно, изображение стало более контрастным.

2.5 Равномерное преобразование

Равномерное преобразование выполняется согласно следующему закону:

$$I_{new} = (I_{max} - I_{min}) * P(I) + I_{min} \quad (2)$$

где I – исходное изображение, I_{new} – преобразованное изображение, I_{min} и I_{max} – минимальное и максимальное значение пикселя на изображении, $P(I)$ – функция распределения вероятностей исходного изображения, которая аппроксимируется коммулятивной гистограммой:

$$P(I) \approx \sum_{m=0}^i Hist(m) \quad (3)$$

Исходный код функции для равномерного преобразования:

```
def uniform_transformation(img):
    b, g, r = cv2.split(img)

    # calculate the histograms
    hist = calc_hist_normalised(img)

    b_min, b_max = b.min(), b.max()
    g_min, g_max = g.min(), g.max()
    r_min, r_max = r.min(), r.max()

    cumulative_histogram_b = np.cumsum(hist[0])
    cumulative_histogram_g = np.cumsum(hist[1])
    cumulative_histogram_r = np.cumsum(hist[2])

    b = np.clip((b_max - b_min) * cumulative_histogram_b[b] + b_min, 0,
                255)
    g = np.clip((g_max - g_min) * cumulative_histogram_g[g] + g_min, 0,
                255)
    r = np.clip((r_max - r_min) * cumulative_histogram_r[r] + r_min, 0,
                255)

    return cv2.merge([b, g, r]).astype(np.uint8)
```

На рисунке 5 представлен результат применения равномерного преобразования к исходному изображению. Как видно, изображение стало более контрастным, при этом коммулятивная

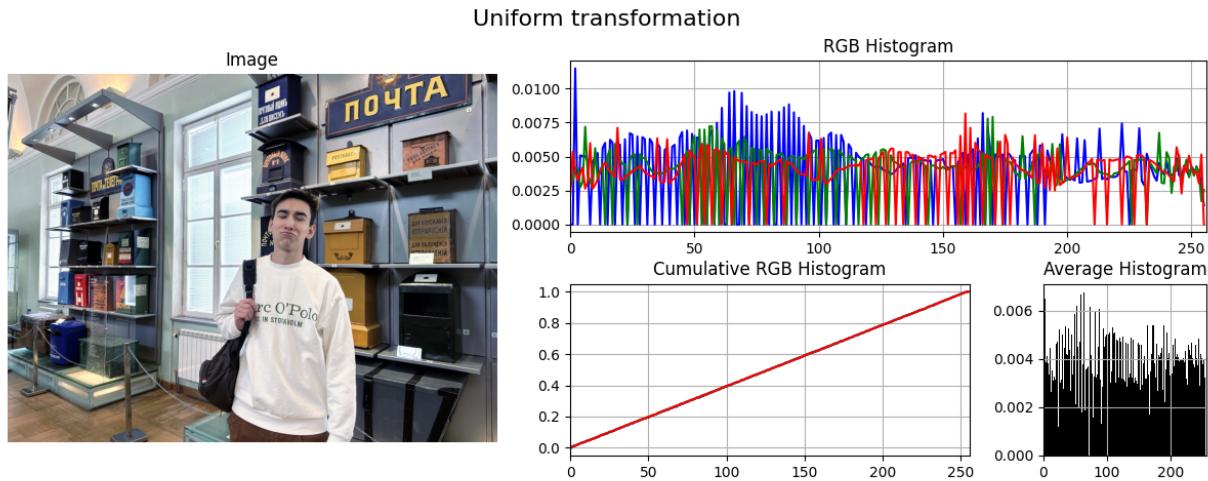


Рис. 5: Результат равномерного преобразования

гистограмма стала представлять из себя линейную функцию, что схоже с результатом применения линейного выравнивания гистограммы.

2.6 Экспоненциальное преобразование

Экспоненциальное преобразование выполняется согласно следующему закону:

$$I_{new} = I_{min} - \frac{1}{\alpha} \cdot \ln(1 - P(I)) \quad (4)$$

где I – исходное изображение, I_{new} – преобразованное изображение, I_{min} – минимальное значение пикселя на изображении, $P(I)$ – функция распределения вероятностей исходного изображения, α – постоянная, характеризующая крутизну преобразования.

Исходный код функции для экспоненциального преобразования:

```
def exponential_transformation(img, alpha):
    b, g, r = cv2.split(img)
    # calculate the histograms
    hist = calc_hist_normalised(img)

    b_min, b_max = b.min(), b.max()
    g_min, g_max = g.min(), g.max()
    r_min, r_max = r.min(), r.max()
```

```

cumulative_histogram_b = np.cumsum(hist[0])
cumulative_histogram_g = np.cumsum(hist[1])
cumulative_histogram_r = np.cumsum(hist[2])

b = b_min - 255/alpha * np.log(1 - cumulative_histogram_b[b])
g = g_min - 255/alpha * np.log(1 - cumulative_histogram_g[g])
r = r_min - 255/alpha * np.log(1 - cumulative_histogram_r[r])

return (cv2.merge([b, g, r])).astype(np.uint8).clip(0, 255)

```

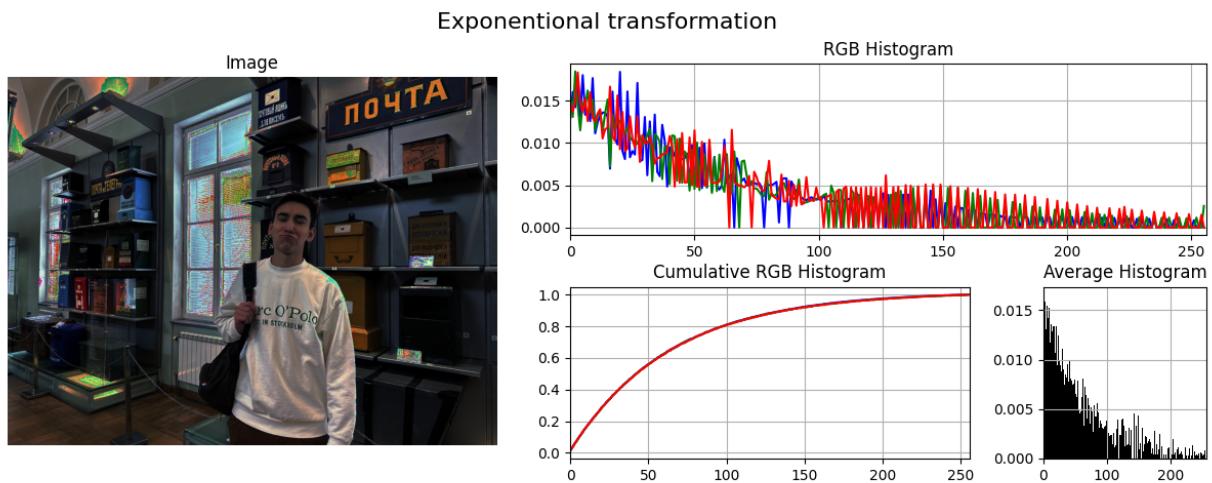


Рис. 6: Результат экспоненциального преобразования

На рисунке 6 видим, что график коммулятивной гистограммы действительно стал представлять из себя экспоненциальную функцию.

При этом изображение стало значительно темнее, что так же видно на гистограммах – темные пиксели стали преобладать на изображении.

2.7 Преобразование по закону Рэлея

Преобразование по закону Рэлея выполняется согласно следующему закону:

$$I_{new} = I_{min} + \left(2\alpha^2 \cdot \ln \left(\frac{1}{1 - P(I)} \right) \right) \quad (5)$$

где α – постоянная, характеризующая гистограмму распределения интенсивностей элементов результирующего изображения.

Исходный код функции для преобразования по закону Рэлея:

```
def rayleigh_law_transformation(img, alpha):  
    b, g, r = cv2.split(img)  
    # calculate the histograms  
    hist = calc_hist_normalised(img)  
  
    b_min, b_max = b.min(), b.max()  
    g_min, g_max = g.min(), g.max()  
    r_min, r_max = r.min(), r.max()  
  
    cumulative_histogram_b = np.cumsum(hist[0])  
    cumulative_histogram_g = np.cumsum(hist[1])  
    cumulative_histogram_r = np.cumsum(hist[2])  
  
    b = np.clip(b_min + (2*alpha**2 * np.log(1 / (1 -  
        cumulative_histogram_b[b]))) ** 0.5 * 255, 0, 255)  
    g = np.clip(g_min + (2*alpha**2 * np.log(1 / (1 -  
        cumulative_histogram_g[g]))) ** 0.5 * 255, 0, 255)  
    r = np.clip(r_min + (2*alpha**2 * np.log(1 / (1 -  
        cumulative_histogram_r[r]))) ** 0.5 * 255, 0, 255)  
  
    return cv2.merge([b, g, r]).astype(np.uint8)
```

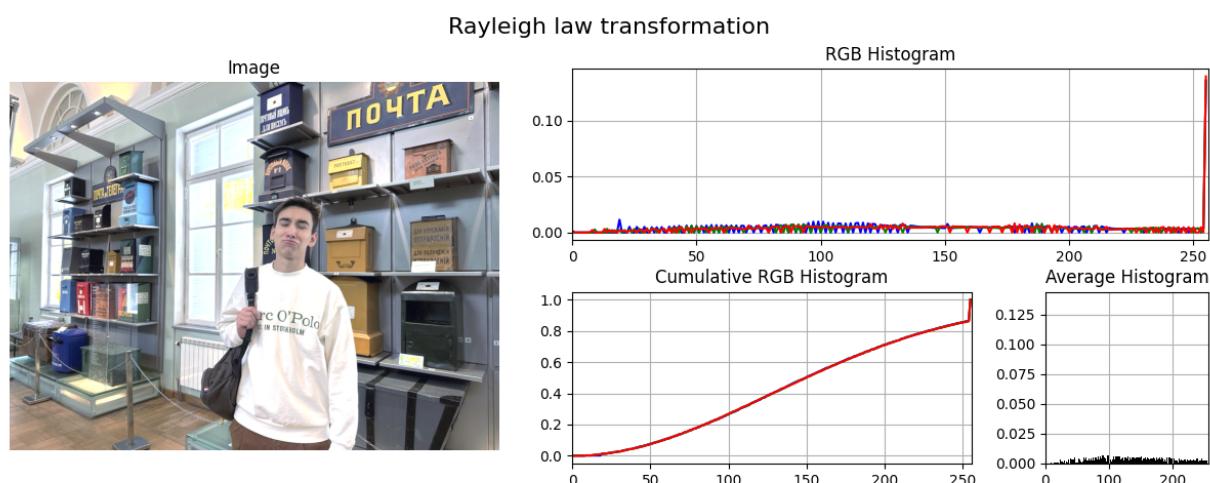


Рис. 7: Результат преобразования по закону Рэлея

На рисунке 7 видим, что график коммулятивной гистограммы действительно стал представлять из себя функцию Рэлея. Так же наблюдаем сильное преобладание светлых пикселей на изображении.

2.8 Преобразование по закону степени 2/3

Преобразование по закону степени 2/3 выполняется согласно следующему закону:

$$I_{new} = P(I)^{\frac{3}{2}} \quad (6)$$

Исходный код функции для преобразования по закону степени 2/3:

```
def two_thirds_rule_transformation(img):
    b, g, r = cv2.split(img)

    # calculate the histograms
    hist = calc_hist_normalised(img)

    b_min, b_max = b.min(), b.max()
    g_min, g_max = g.min(), g.max()
    r_min, r_max = r.min(), r.max()

    cumulative_histogram_b = np.cumsum(hist[0])
    cumulative_histogram_g = np.cumsum(hist[1])
    cumulative_histogram_r = np.cumsum(hist[2])

    b = np.clip((cumulative_histogram_b[b] ** (2/3) * 255), 0, 255)
    g = np.clip((cumulative_histogram_g[g] ** (2/3) * 255), 0, 255)
    r = np.clip((cumulative_histogram_r[r] ** (2/3) * 255), 0, 255)

    return cv2.merge([b, g, r]).astype(np.uint8)
```

На рисунке 8 видим, что график коммулятивной гистограммы действительно стал представлять из себя функцию степени 2/3.

изображение стало намного более светлым, контрастность упала.



Рис. 8: Результат преобразования по закону степени 2/3

2.9 Гиперболическое преобразование

Гиперболическое преобразование выполняется согласно следующему закону:

$$I_{new} = \alpha^{P(I)} \quad (7)$$

где α – постоянная, относительно которой осуществляется преобразование.

Исходный код функции для гиперболического преобразования:

```
def hyperbolic_transformation(img, alpha):
    b, g, r = cv2.split(img)

    # calculate the histograms
    hist = calc_hist_normalised(img)

    b_min, b_max = b.min(), b.max()
    g_min, g_max = g.min(), g.max()
    r_min, r_max = r.min(), r.max()

    cumulative_histogram_b = np.cumsum(hist[0])
    cumulative_histogram_g = np.cumsum(hist[1])
    cumulative_histogram_r = np.cumsum(hist[2])

    b = np.clip(alpha ** cumulative_histogram_b[b] * 255, 0, 255)
    g = np.clip(alpha ** cumulative_histogram_g[g] * 255, 0, 255)
```

```

r = np.clip(alpha ** cumulative_histogram_r[r] * 255, 0, 255)

return cv2.merge([b, g, r]).astype(np.uint8)

```

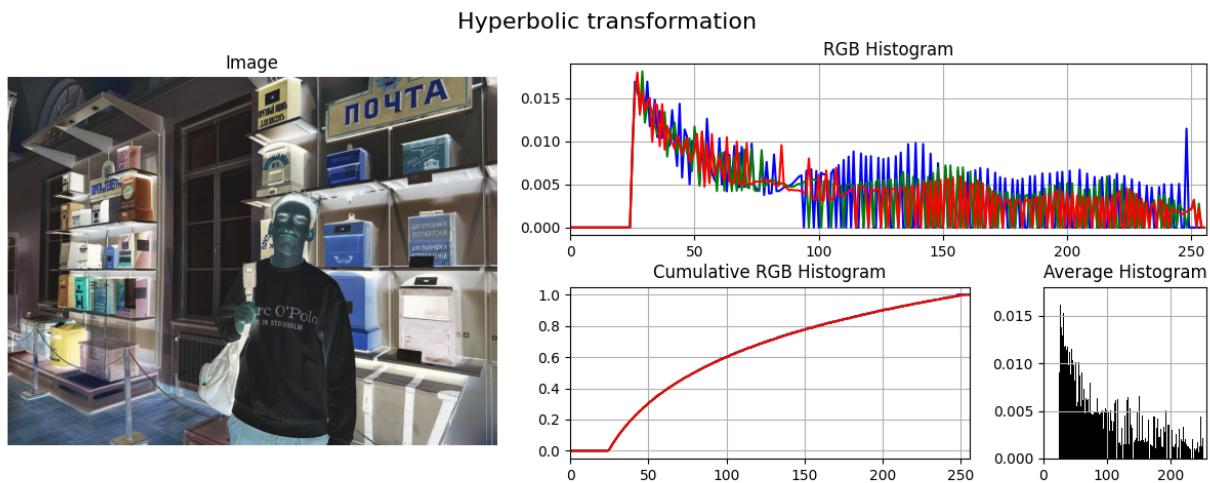


Рис. 9: Результат гиперболического преобразования

На рисунке 9 видим, что график коммулятивной гистограммы действительно стал представлять из себя гиперболу.

Изображение можно описать как негативное от исходного, заметно явное преобладание темных пикселей на изображении.

2.10 Таблица поиска

Таблица поиска (lookup table) – это таблица, в которой для каждого значения пикселя изображения задается новое значение пикселя.

Исходный код функции для преобразования по таблице поиска:

```

def LUT_transformation(img, LUT):
    return cv2.LUT(img, LUT).astype(np.uint8)

```

Для создания самой таблицы поиска использовался следующий код:

```

# generate LUT

lut = np.arange(256, dtype = np.uint8)
lut = np.clip(np.power(lut, 0.9) + 20, 0, 255)

```

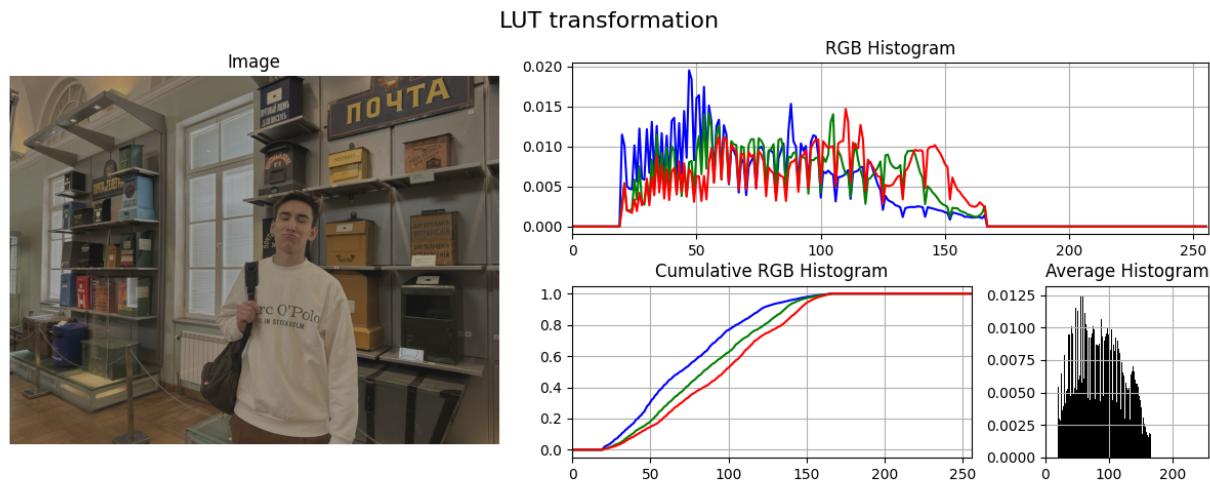


Рис. 10: Результат преобразования по таблице поиска

На рисунке 10 видим, что график коммулятивной гистограммы стал представлять из себя функцию степени 0.9, смешение на 20, что и было задано в таблице поиска.

3 Прифили изображения

Прифилем изображения вдоль некоторой линии называется функция интенсивности изображения, распределенного вдоль этой линии (*прорезки*).

Рассмотрим профили изображения вдоль горизонтальной и вертикальной линий:

$$\text{Profile } i(x) = I(x, i) \quad (8)$$

$$\text{Profile } i(y) = I(i, y) \quad (9)$$

где $I(x, i)$ – интенсивность пикселя изображения в точке (x, i) , $I(i, y)$ – интенсивность пикселя изображения в точке (i, y) .

3.1 Профиль изображения вдоль горизонтальной линии

Рассмотрим профильт изображения со штрих-кодом вдоль горизонтальной линии.

Исходный код функции для вычисления профиля изображения вдоль горизонтальной линии:

```
def show_image_profile(img, level):
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # normal colors to
    display
    plt.figure(figsize=(13, 5))
    plt.subplot(1, 2, 1)
    plt.title('Image')
    plt.imshow(img_rgb)
    plt.axis('off') # disable the axis

    plt.subplot(1, 2, 2)
    profile = img[level, :]
    plt.plot(profile)
    plt.title('Profile')
```

```

plt.subplots_adjust(wspace=0.3, hspace=0.3)
plt.subplots_adjust(left=0.05, right=0.95)

show_image_profile(img, img.shape[0] // 2)

```

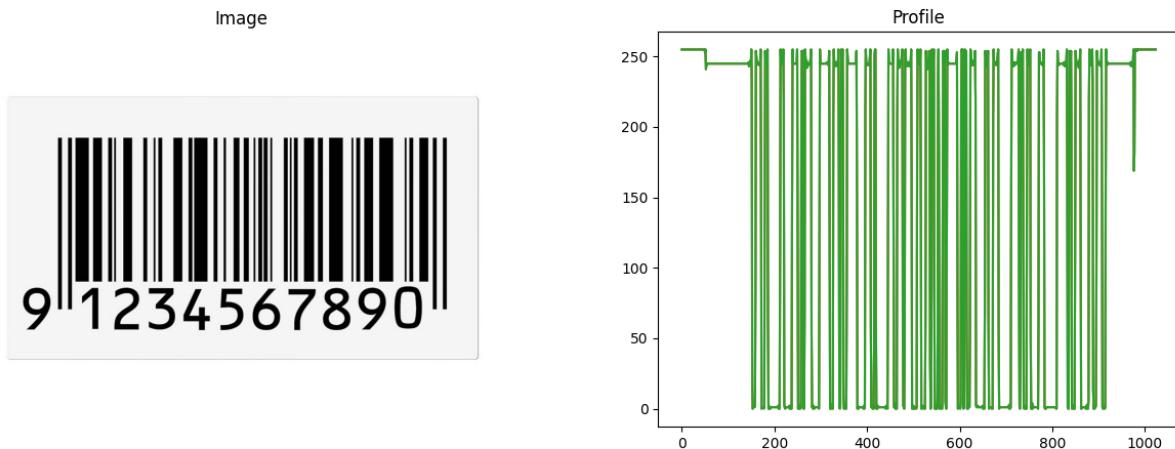


Рис. 11: Профиль изображения вдоль горизонтальной линии

Видим, что на проекции хорошо различимы *штрихи* штрих-кода. Данную проекцию можно использовать для распознавания штрих-кода.

4 Проекция изображения

Проекцией изображения на некоторую ось называется сумма интенсивностей пикселей изображения в направлении, перпендикулярном этой оси.

4.1 Проекция изображения на оси

Рассмотрим проекцию изображения на горизонтальную и вертикальную оси:

$$\text{Projection } X(y) = \sum_{i=0}^{\dim Y - 1} I(y, i) \quad (10)$$

$$\text{Projection } Y(x) = \sum_{i=0}^{\dim X - 1} I(i, x) \quad (11)$$

Исходный код функции для вычисления проекции изображения на оси:

```
def show_image_projection(img, title="Projection"):  
    b, g, r = cv2.split(img)  
  
    # calculate Ox projection  
    projection_x = (np.sum(b, axis=0) + np.sum(g, axis=0) + np.sum(r,  
        axis=0)) / img.shape[0] / 3  
  
    # calculate Oy projection  
    projection_y = (np.sum(b, axis=1) + np.sum(g, axis=1) + np.sum(r,  
        axis=1)) / img.shape[1] / 3  
  
    plt.figure(figsize=(8, 8))  
    plt.subplot(2, 2, 1)  
    plt.grid(True)  
    plt.title('Image')  
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))  
  
    plt.subplot(2, 2, 3)  
    plt.ylim([255, 0])  
    plt.xlim([0, img.shape[1]])  
    plt.grid(True)  
    plt.plot(range(img.shape[1]), projection_x, color='g')  
    plt.title('Projection X')  
  
    plt.subplot(2, 2, 2)  
    plt.xlim([0, 255])  
    plt.ylim([img.shape[1], 0])  
    plt.grid(True)  
    plt.plot(projection_y, range(img.shape[0]), color='g')  
    plt.title('Projection Y')  
  
    plt.subplots_adjust(wspace=0.3, hspace=0.3)  
    plt.subplots_adjust(left=0.05, right=0.95)  
  
    # save image  
    plt.savefig(f"results/{title}.png")
```

```

show_image_projection(img1, "Projection 1")
show_image_projection(img2, "Projection 2")

```

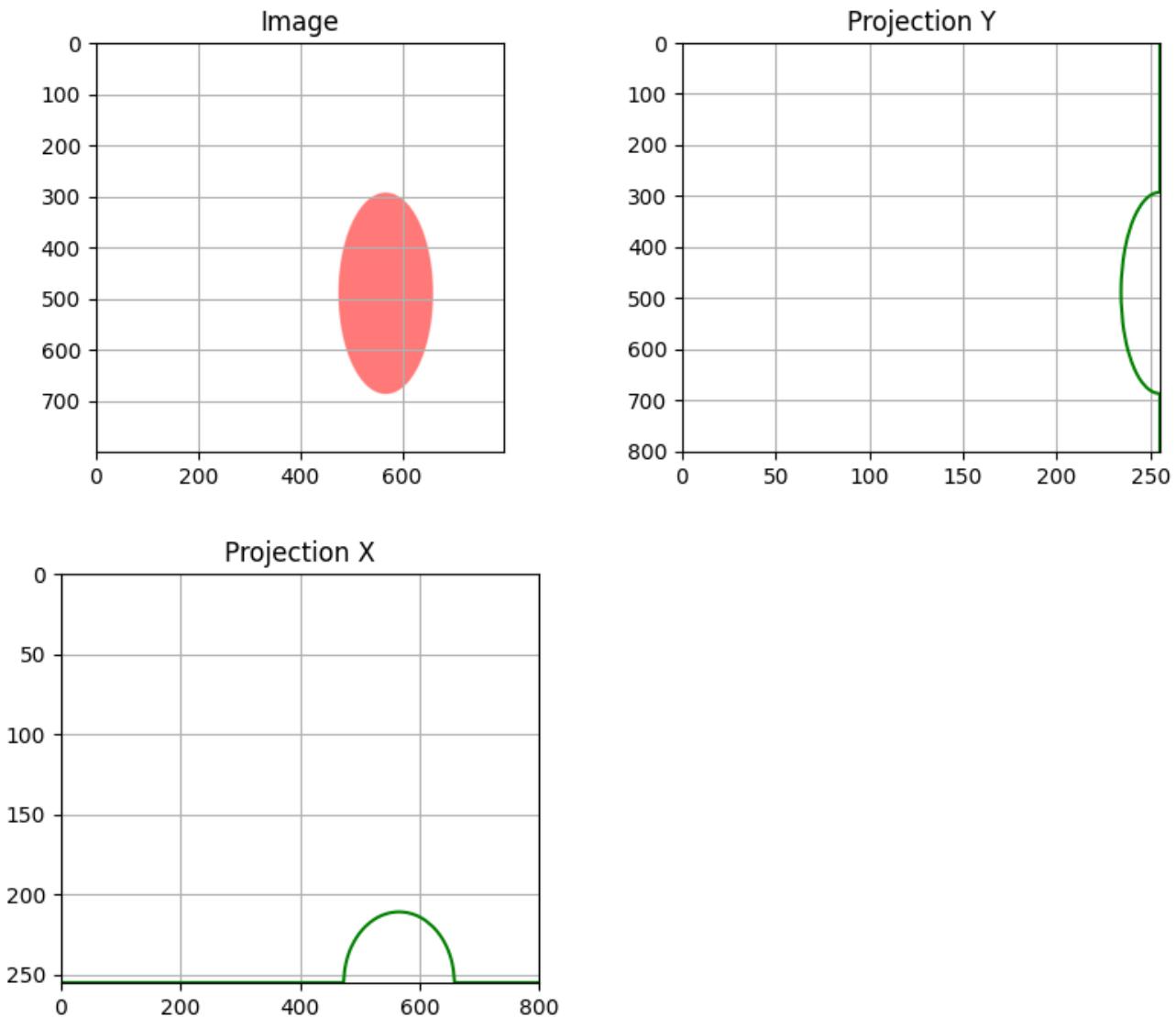


Рис. 12: Проекция изображения 1

На рисунках 12 и 13 представлены проекции изображений на горизонтальную и вертикальную оси. По этим проекциям можно определить расположение объектов на изображении, а также их размеры.

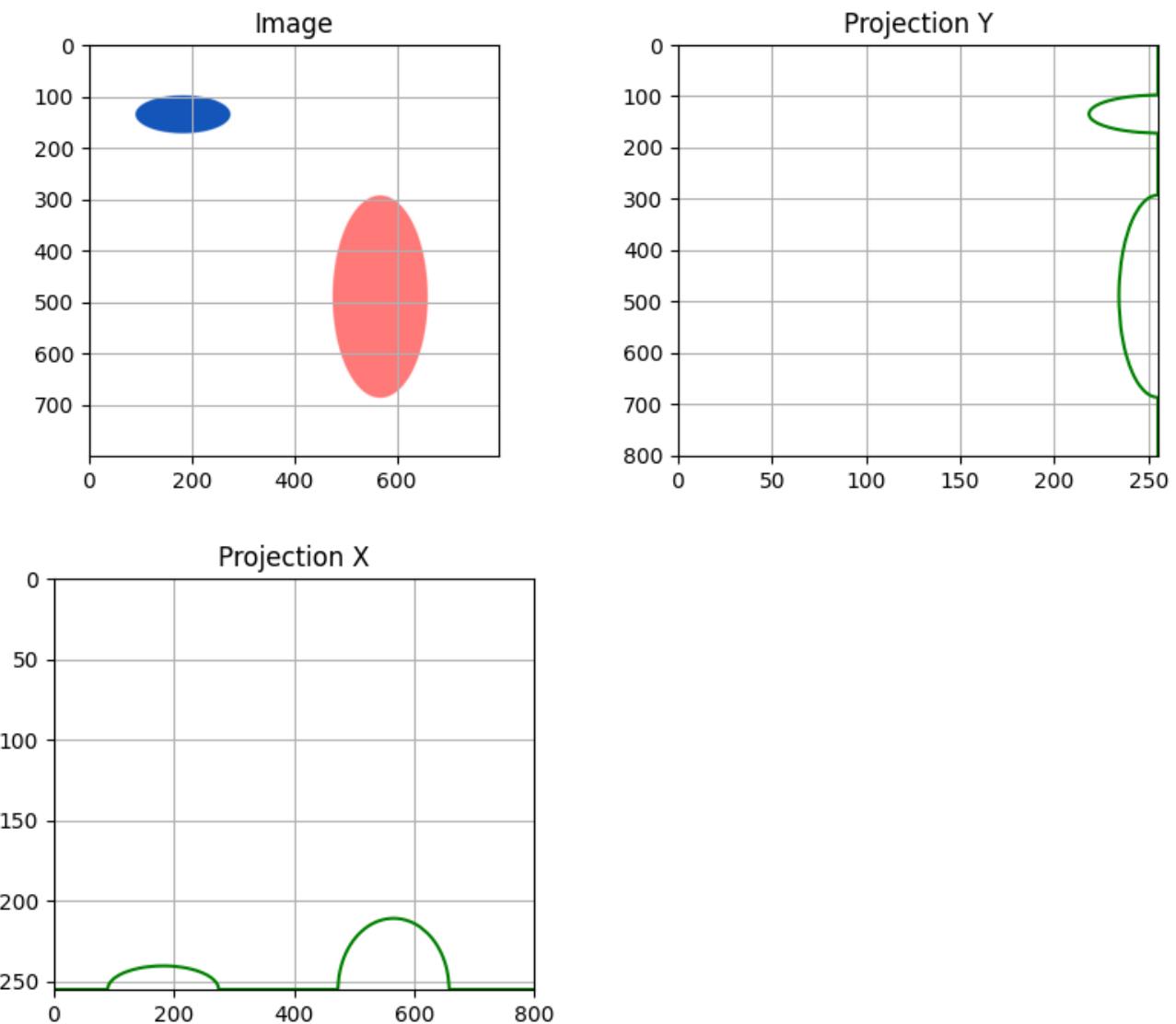


Рис. 13: Проекция изображения 2

5 Вывод

Exercitation non cupidatat exercitation velit nostrud dolor. Ipsum dolor elit ex et minim enim. Occaecat occaecat eu eu cillum pariatur velit non anim culpa. Esse officia magna duis aliqua dolore. In anim commodo consectetur ullamco fugiat Lorem magna nisi.