

Федеральное государственное автономное образовательное
учреждение высшего образования

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет систем управления и робототехники

Лабораторная работа №1
ГИСТОГРАММЫ, ПРОФИЛИ И ПРОЕКЦИИ

Студенты: Овчинников П.А.
Румянцев А.А.
Чебаненко Д.А.

Поток: ТЕХ.ЗРЕНИЕ 2.1

Преподаватель: Шаветов С.В.

Санкт-Петербург
2024

Содержание

Цель работы	3
Код, используемый в работе	3
Вычисление гистограмм	3
Отрисовка гистограмм и изображения	3
Преобразование гистограммы изображения	5
Линейное выравнивание гистограммы	6
Арифметические операции	7
Растяжение динамического диапазона	8
Равномерное преобразование	9
Экспоненциальное преобразование	10
Преобразование по закону Рэлея	11
Преобразование по закону степени $2/3$	12
Гиперболическое преобразование	13
Таблица поиска	14
Профили и проекции	15
Профиль изображения вдоль горизонтальной линии по середине	15
Проекция изображения на оси	16
Вывод	17
Вопросы к защите	17

Цель работы

В этой работе мы изучим основные яркостные и геометрические характеристики изображений, чтобы использовать их для анализа изображения. Мы научимся вычислять гистограммы, профили и проекции изображений, чтобы в дальнейшем использовать их в других задачах вне лабораторной работы.

Код, используемый в работе

Для начала стоит отметить, что весь код опубликован в [приватном гисте в GitHub](#), доступном только по ссылке из этого отчёта. Часть кода будет приведена под этим заголовоком с указанием необходимых импортов, и остальная часть будет приведена в соответствующих разделах отчёта уже без указания импортов для экономии места.

Вычисление гистограмм

```

1 from typing import Sequence
2 import cv2
3 from cv2.typing import MatLike
4 import numpy as np
5 from numpy._typing import NDArray
6
7
8 def calc_rgb_hist(channels: Sequence[MatLike]) -> list[MatLike]:
9     """Вычисляет нормализованные гистограммы изображения"""
10    hists = []
11    for i in range(3):
12        hist = cv2.calcHist(channels, [i], None, [hist_size], hist_range)
13        hists.append(hist / (image.shape[0] * image.shape[1])) # Нормализуем гистограмму
14    return hists
15
16
17 def calc_cumulative_hist(channels: Sequence[MatLike]) -> tuple[NDArray, ...]:
18     """Вычисляет кумулятивную гистограмму изображения"""
19     # Подсчитываем гистограммы по каналам, применяем к ним np.cumsum и возвращаем в кортеже
20     return tuple(map(np.cumsum, calc_rgb_hist(channels)))

```

Листинг 1: Вычисление нормализованных RGB- и кумулятивных гистограмм

Перед вами две функции, одна из которых подсчитывает нормализованную RGB-гистограмму, а вторая использует первую, чтобы подсчитать кумулятивную гистограмму по каждому из каналов. В обоих случаях используется библиотека OpenCV для работы с изображениями и библиотека NumPy для работы с массивами. Функция, вычисляющая кумулятивную гистограмму, будет активно использоваться в течение всей лабораторной работы.

Отрисовка гистограмм и изображения

```

1 from typing import Callable, Sequence
2 import cv2
3 from cv2.typing import MatLike
4 import matplotlib.pyplot as plt
5 import numpy as np
6
7
8 def clear():
9     """Очищает холст matplotlib от графиков"""
10    plt.cla() # Очищаем график
11    plt.clf() # Очищаем фигуру
12    plt.subplots_adjust(left=0.07, right=0.95)
13
14
15 def render_rgb_hist(channels: Sequence[MatLike], name: str):
16     """Рендерит гистограмму изображения из нормализованных гистограмм каждого канала"""
17    clear()
18    hists = calc_rgb_hist(channels)
19
20    plt.grid(True)

```

```

21     plt.xlim(hist_range)
22     plt.ylim(-0.001, 0.1)
23     for hist, color in zip(hists[::-1], ('r', 'g', 'b')):
24         plt.plot(hist, color=color)
25
26 plt.title('RGB-гистограмма')
27 plt.savefig(f'{name}.png')
28
29
30 def render_cumulative_hist(channels: Sequence[MatLike], name: str):
31     """Рендерит кумулятивную гистограмму изображения"""
32     clear()
33     cumhists = calc_cumulative_hist(channels)
34
35     plt.grid(True)
36     plt.xlim(hist_range)
37     plt.ylim(-0.005, 1.005)
38     for hist, color in zip(cumhists[::-1], ('r', 'g', 'b')):
39         plt.plot(hist, color=color)
40
41 plt.title('Кумулятивная гистограмма')
42 plt.savefig(f'{name}.png')
43
44
45 def apply_method(method: Callable, image: MatLike, output_dir: str, **kwargs):
46     """Применяет преобразование к изображению, компрессирует значения выше
47     максимальной яркости до 255 и сохраняет результат в папку output_dir"""
48     new_image = method(image, **kwargs)
49     cv2.threshold(new_image, 255, 255, cv2.THRESH_TRUNC, new_image)
50     new_image = new_image.astype(np.uint8)
51     cv2.imwrite(f'{output_dir}/photo.jpg', new_image)
52
53 # Отрендерим гистограммы преобразованного изображения
54 render_rgb_hist(cv2.split(new_image), f'{output_dir}/hist')
55 render_cumulative_hist(cv2.split(new_image), f'{output_dir}/cumhist')

```

Листинг 2: Отрисовка RGB- и кумулятивной гистограмм

Этот код отвечает за отрисовку гистограмм и изображений. В нём используется библиотека `matplotlib` для отрисовки графиков. В функции `clear` происходит очистка холста от предыдущих графиков, чтобы не возникало накладывания одних графиков на другие. Функции `render_rgb_hist` и `render_cumulative_hist` отвечают за отрисовку RGB- и кумулятивной гистограмм соответственно. В обоих случаях используются функции для посчёта гистограмм, которые были определены в предыдущем листинге.

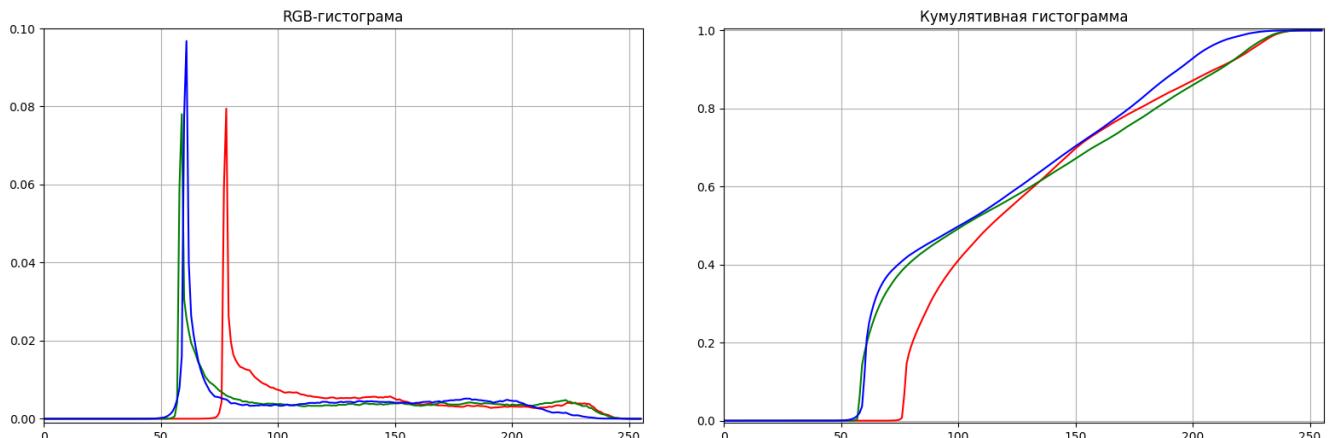
Отдельного внимания заслуживает функция `apply_method`, которая принимает на вход функцию `method`, изображение `image`, папку для сохранения результата и дополнительные аргументы, которые передаются вместе с изображением в `method`. Затем функция применяет преобразование к изображению, компрессирует значения выше максимальной яркости до 255 (это нужно для того, чтобы избежать искажений, связанных с циклическим повторением гистограммы) и сохраняет результат в папку `output_dir`. После этого она вызывает функции для отрисовки гистограмм уже преобразованного изображения, чтобы можно было сравнить гистограммы.

Преобразование гистограммы изображения

Перед началом работы с преобразованиями стоит продемонстрировать исходное изображение и его гистограммы.



Рис. 1: Исходное изображение



Итак, вот гистограммы исходного изображения, из которых можно сделать такие выводы:

- На RGB-гистограмме видны три пика для каждого канала в тёмной части спектра
- Градации до 50-ти вовсе отсутствуют на изображении
- Изображение в общем и целом не очень яркое, что хорошо видно по обеим гистограммам на градациях выше 100.

Очевидно, это изображение хочет, чтобы мы навели в нём красоту, а значит мы готовы преобразовываться! Для каждого из преобразований будет приведён код функции, которая реализует его, а также код для запуска преобразования с помощью функции `apply_method` для понимания того, какие коэффициенты передаются в функцию преобразования.

Линейное выравнивание гистограммы

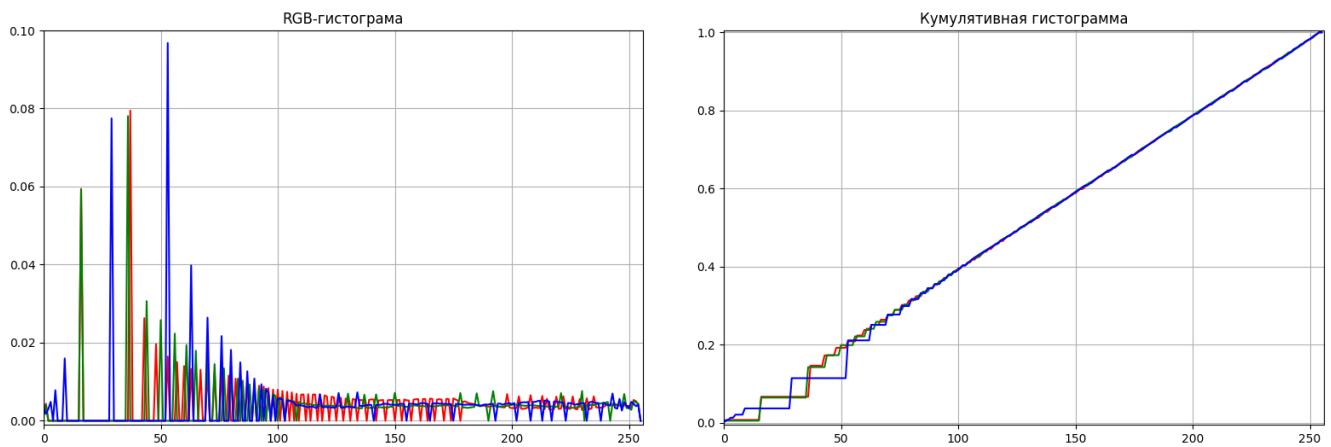
Линейное выравнивание гистограммы требует наличия кумулятивной гистограммы, которая в свою очередь будет выравнена этим методом в прямую линию. Ниже код для этого преобразования:

```

1 def linear_alignment(image: MatLike) -> NDArray[np.uint8]:
2     """Линейное выравнивание гистограммы"""
3     channels = cv2.split(image)
4     cumhists = calc_cumulative_hist(channels)
5     new_channels = [] # Будем формировать новые каналы в этот список
6     for channel, cumhist in zip(channels, cumhists):
7         # Применяем линейное выравнивание к каждому каналу по кумулятивной гистограмме
8         # В гистограмме диапазон [0..1], поэтому умножаем на 255
9         new_channels.append(255 * cumhist[channel])
10    return cv2.merge(new_channels)
11
12
13 apply_method(linear_alignment, image, 'aligned')

```

Листинг 3: Код, реализующий линейное выравнивание гистограммы



И мы видим, что RGB-гистограмма изображения изменилась так, что теперь кумулятивная гистограмма выглядит как равномерная линия (в начале гистограммы наблюдаются скачки, связанные с низким качеством изображения). В целом, это преобразование натурализировало цвета изображения, избавив его от красноватого оттенка, и сделало его ярче. К тому же почти не заметна тёмная виньетка по краям изображения. И, самое главное, преобразование сделало это без засветов на небе — мы всё ещё видим детали в облаках, это говорит о повышении контрастности.

Арифметические операции

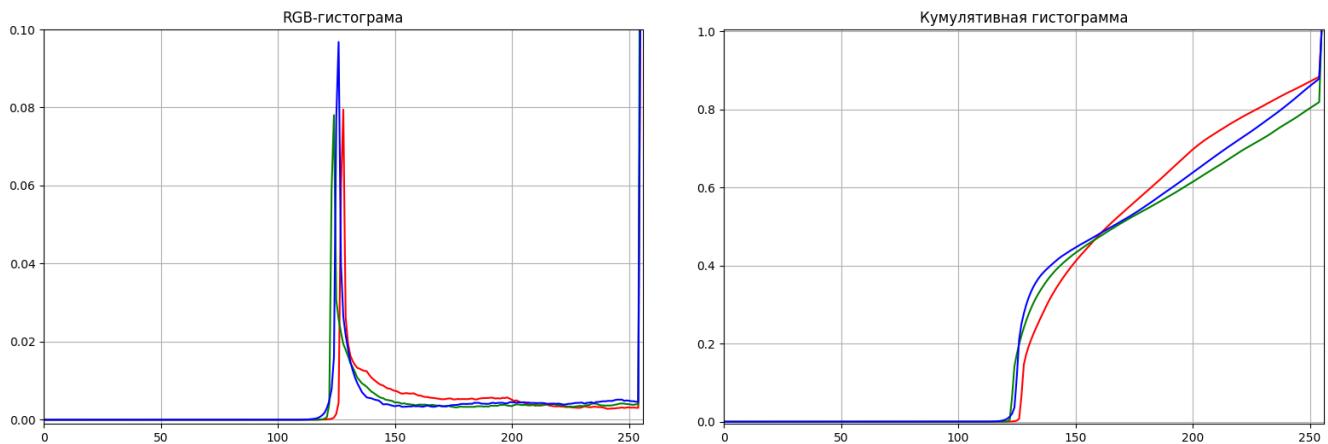
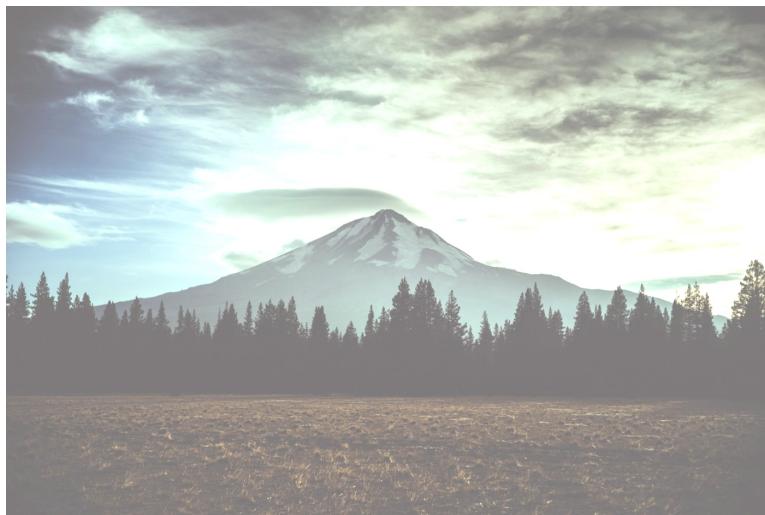
Одно из простейших преобразований — **арифметические операции**, т.е. добавление и вычитание какого-то количества градаций к каждому из каналов. Исходя из RGB-гистограммы первоначального изображения, устанавливаем сдвиг синего и зелёного канала на 65 градаций вправо, а сдвиг красного канала — на 50 градаций вправо. Вот код для этого преобразования:

```

1 def arithmetic_operations(image: MatLike, shifts: list[float]) -> NDArray[np.uint8]:
2     """Арифметические операции с изображением"""
3     channels = list(cv2.split(image))
4     for i, channel in enumerate(channels):
5         # Сдвигаем каждый канал на значение shift
6         channels[i] = np.clip(channel.astype(np.uint16) + shifts[i], 0, 255).astype(np.uint8)
7     return cv2.merge(channels)
8
9
10 apply_method(arithmetic_operations, image, '2shift', shifts=[65, 65, 50])

```

Листинг 4: Код, реализующий арифметические операции над изображением



NB: в коде происходит расширение типа данных до `uint16` перед выполнением сдвига, чтобы избежать переполнения с циклическим сдвигом RGB-гистограммы — так мы не будем наблюдать искажения в изображении.

Как мы видим, оттенок в изображении изменился и оно стало ярче, однако применение арифметических операций хоть и простое, но не прибавляет к изображению контрастности и не улучшает визуально — мы потеряли часть изображения в светлых участках, и теперь небо выглядит как однородная масса. Так что это преобразование не слишком практичное.

Растяжение динамического диапазона

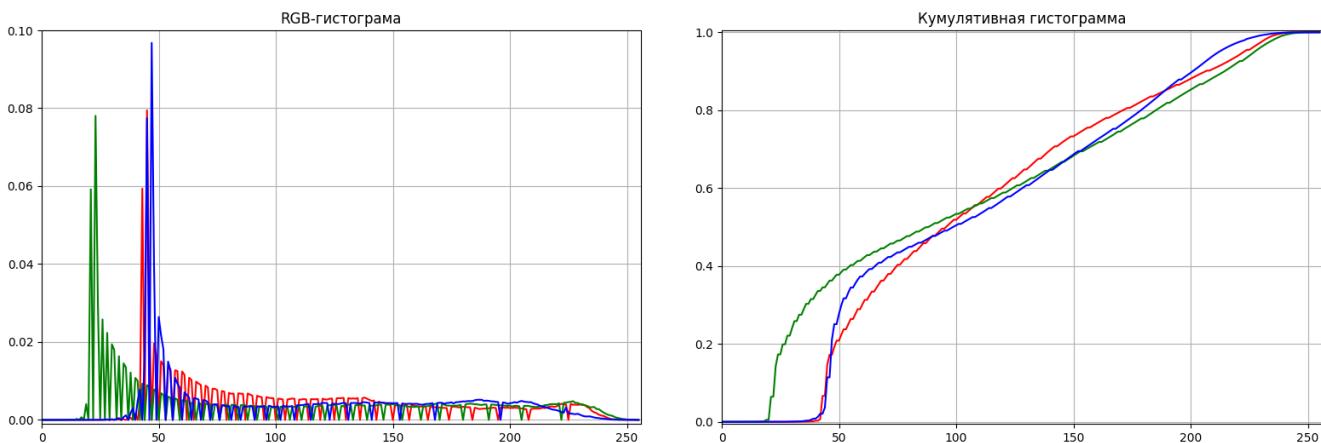
Растяжение динамического диапазона — отличное преобразование, которое расширяет сжатый диапазон до нуля и максимума с обоих сторон гистограммы. Нелинейность преобразования (которую мы заметим на кумулятивной гистограмме) регулируется с помощью коэффициента нелинейности $\alpha = 0.8$. И код перед вами:

```

1 def contrast_stretching(image: MatLike, alpha: float) -> NDArray[np.uint8]:
2     """Расширение контраста методом растяжения динамического диапазона"""
3     # Если изображение в целочисленном диапазоне [0..255], то конвертируем в вещественный [0..1]
4     image = image.astype(np.float64) / 255 if image.dtype == 'uint8' else image
5     channels = list(cv2.split(image))
6     for i, channel in enumerate(channels):
7         cmin, cmax = channel.min(), channel.max()
8         channels[i] = ((channel - cmin) / (cmax - cmin)) ** alpha * 255
9     image = cv2.merge(channels)
10    # Если изображение было в целочисленном диапазоне, то возвращаем его обратно в этот диапазон
11    return (image.clip(0, 255) if image.dtype == 'uint8' else image).astype(np.uint8)
12
13
14 apply_method(contrast_stretching, image, '3stretched', alpha=0.8)

```

Листинг 5: Код, растягивающий динамический диапазон изображения



Как мы видим, с изображения спала белая пелена выгорания, которая добавляла лишнюю яркость всему изображению — из-за этого на небе видно больше деталей и в целом изображение стало более контрастным и сочным в цветах. Однако из-за нелинейности преобразования мы потеряли детали в тёмных участках изображения, и теперь они выглядят как однородная масса. Преобразование подойдёт, чтобы сделать старые снимки с выгораниями лучше и убавить блики, но оно не очень хорошо работает на тёмных изображениях.

Равномерное преобразование

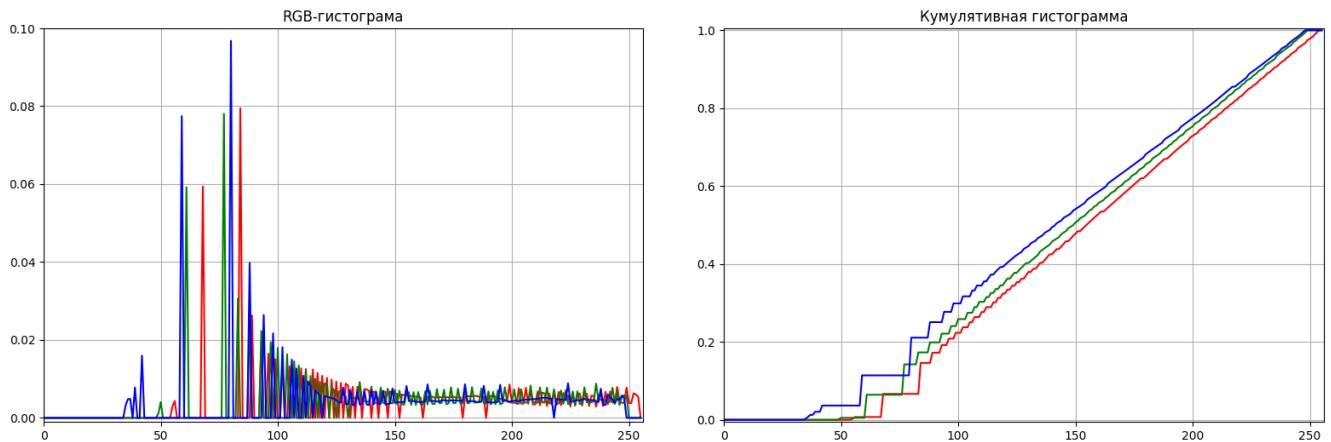
Равномерное преобразование отличается от линейного выравнивания тем, что оно не растягивает гистограмму в прямую линию, а делает её равномерной с самого тёмного пикселя каждого канала. Вот код для этого преобразования:

```

1 def uniform_conversion(image: MatLike) -> NDArray[np.uint8]:
2     """Равномерное преобразование гистограммы"""
3     channels = cv2.split(image)
4     cumhists = calc_cumulative_hist(channels)
5     new_channels = []
6     for channel, cumhist in zip(channels, cumhists):
7         cmin, cmax = channel.min(), channel.max()
8         new_channels.append((cmax - cmin) * cumhist[channel] + cmin)
9     return cv2.merge(new_channels)
10
11
12 apply_method(uniform_conversion, image, 'uniform')

```

Листинг 6: Код, реализующий равномерное преобразование



Изображение стало более контрастным, и теперь на нём видно больше деталей в тёмных участках, но при этом яркость изображения не изменилась и по-прежнему наблюдается выгорание, однако тёмная виньетка вокруг изображения практически полностью устранена. Преобразование идеально подойдёт для тёмных изображений, но не для ярких, т.к. яркость изображения не меняется.

Экспоненциальное преобразование

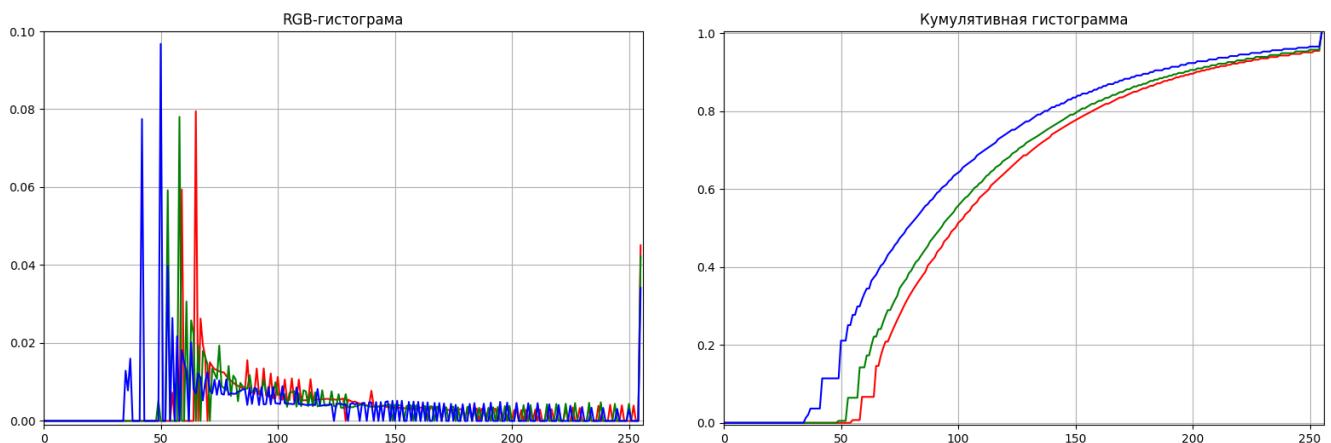
Экспоненциальное преобразование — это преобразование, которое подгоняет RGB-гистограмму так, что кумулятивная гистограмма становится экспоненциальной функцией, крутизну которой можно регулировать с помощью коэффициента $\alpha = 4$. Давайте взглянем на код:

```

1 def exponential_conversion(image: MatLike, alpha: float) -> NDArray[np.uint8]:
2     """Экспоненциальное преобразование гистограммы"""
3     channels = cv2.split(image)
4     cumhists = calc_cumulative_hist(channels)
5     new_channels = []
6     for channel, cumhist in zip(channels, cumhists):
7         cmin = channel.min()
8         new_channels.append(cmin - 255 / alpha * np.log(1 - cumhist[channel]))
9     return cv2.merge(new_channels)
10
11
12 apply_method(exponential_conversion, image, '5_exponential', alpha=4)

```

Листинг 7: Код, реализующий экспоненциальное преобразование



Изображение стало гораздо темнее, но при этом детали елей ещё различимы, а небо выглядит более детализированным, как и трава на поле. При этом виньетка вокруг изображения стала меньше, а на небе видны яркие пятна. Преобразование хорошо подойдёт для тёмных изображений с широким диапазоном.

Преобразование по закону Рэлея

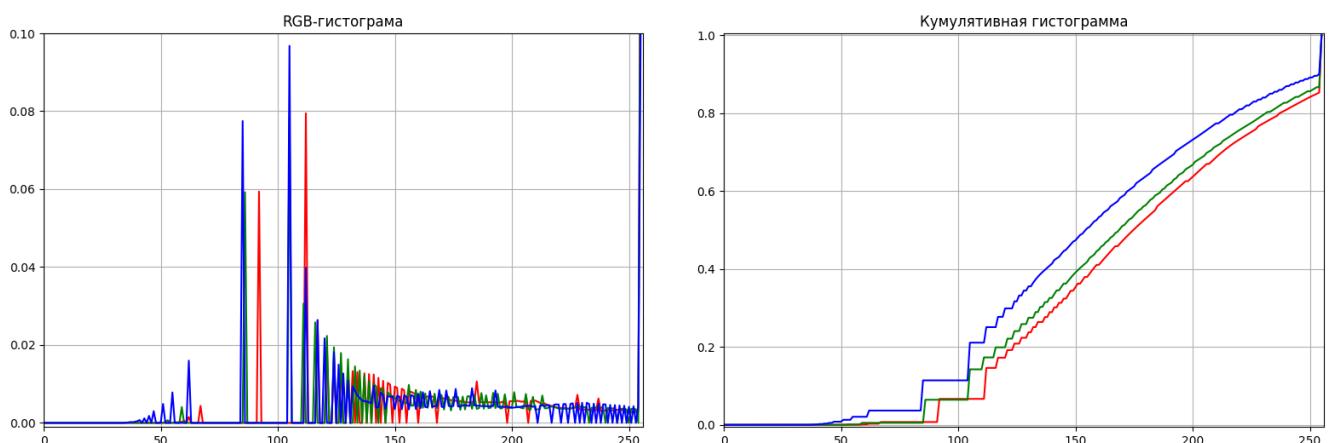
Преобразование по закону Рэлея — это преобразование, которое подгоняет RGB-гистограмму так, что кумулятивная гистограмма становится функцией Рэлея, причём угол наклона функции можно регулировать с помощью коэффициента $\alpha = 0.4$. А теперь посмотрим на код этого преобразования:

```

1 def rayleigh_conversion(image: MatLike, alpha: float) -> NDArray[np.uint8]:
2     """Преобразование гистограммы по закону Рэлея"""
3     channels = cv2.split(image)
4     cumhists = calc_cumulative_hist(channels)
5     new_channels = []
6     for channel, cumhist in zip(channels, cumhists):
7         cmin = channel.min()
8         new_channels.append(cmin + np.sqrt(2 * alpha**2 * np.log(1 / (1 - cumhist[channel])))) * 255
9     return cv2.merge(new_channels)
10
11
12 apply_method(rayleigh_conversion, image, '6rayleigh', alpha=0.4)

```

Листинг 8: Код, реализующий преобразование по закону Рэлея



Сразу видно, что изображение стало сильно ярче, при этом много деталей на светлых участках неба упущено. Но тёмные участки изображения стали более контрастными, что отличает преобразование по закону Рэлея от обычных арифметических операций, лишь изменяющих яркость изображение. Преобразование подойдёт для тёмных изображений с узким диапазоном, но не для ярких изображений и не для широкого диапазона.

Преобразование по закону степени $2/3$

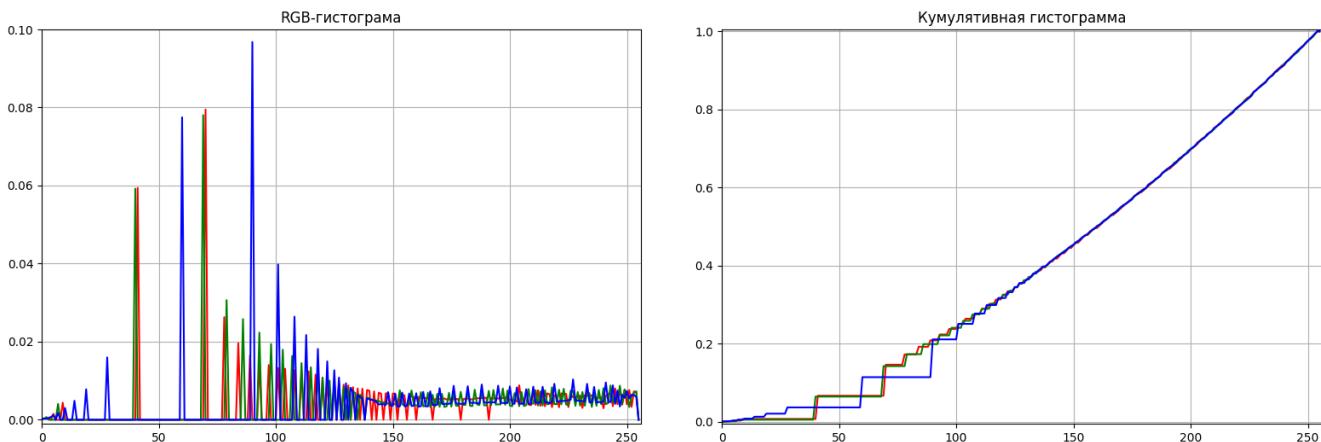
Преобразование по закону степени $2/3$ — это преобразование, которое подгоняет RGB-гистограмму так, что кумулятивная гистограмма становится показательной функцией степени $2/3$. Рассмотрим код этого преобразования:

```

1 def rule23_conversion(image: MatLike) -> NDArray[np.uint8]:
2     """Преобразование гистограммы по закону степени 2/3"""
3     channels = cv2.split(image)
4     cumhists = calc_cumulative_hist(channels)
5     new_channels = []
6     for channel, cumhist in zip(channels, cumhists):
7         new_channels.append(cumhist[channel] ** (2 / 3) * 255)
8     return cv2.merge(new_channels)
9
10
11 apply_method(rule23_conversion, image, 'rule23')

```

Листинг 9: Код, реализующий преобразование по закону степени $2/3$



Преобразование очень похоже на линейное выравнивание, но при этом оно более нелинейное, что добавляет в яркости и в контрастности, но вносит небольшие искажения в тёмные участки. Это преобразование подойдёт к более тёмным и детализированным изображениям, но не к ярким и с высокой контрастностью.

Гиперболическое преобразование

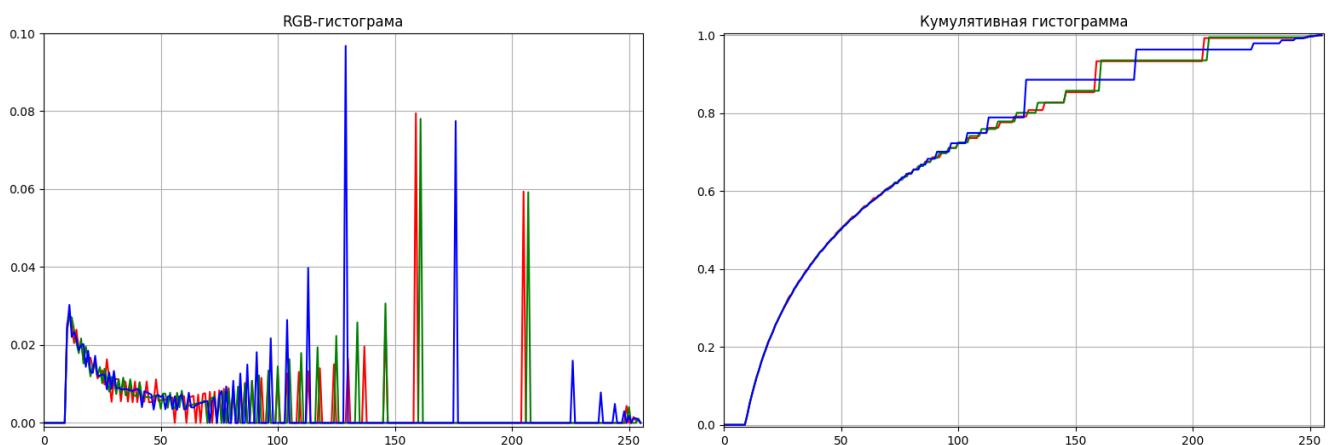
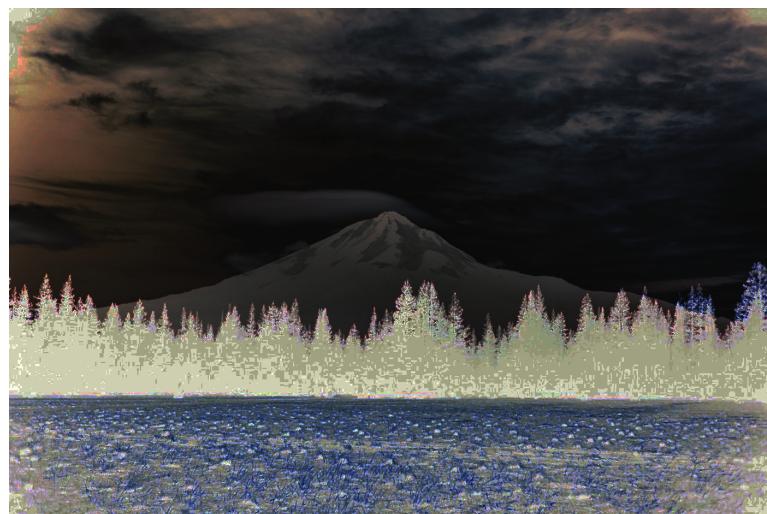
Гиперболическое преобразование — это преобразование, которое подгоняет RGB-гистограмму так, что кумулятивная гистограмма становится похожа на гиперболу в четвёртой четверти декартовой системы координат. Ввиду того, что в формуле показательная функция, в отличие от, например, предыдущего преобразования, мы получим довольно неожиданный результат. И вот как выглядит код этого преобразования:

```

1 def hyperbolic_conversion(image: MatLike, alpha: float) -> NDArray[np.uint8]:
2     """Преобразование гистограммы по закону степени 2/3"""
3     channels = cv2.split(image)
4     cumhists = calc_cumulative_hist(channels)
5     new_channels = []
6     for channel, cumhist in zip(channels, cumhists):
7         new_channels.append(alpha ** cumhist[channel] * 255)
8     return cv2.merge(new_channels)
9
10
11 apply_method(hyperbolic_conversion, image, 'hyperbolic', alpha=0.04)

```

Листинг 10: Код, реализующий гиперболическое преобразование



Глубину получившегося негатива можно регулировать, изменяя $\alpha = 0.04$ — этот коэффициент влияет на крутизну гиперболы и на яркость и контраст изображения. Такой негатив подойдёт для отделения фона от объекта на изображении и для его анализа, но, конечно, здесь не идёт речи об улучшении визуального качества изображения.

Таблица поиска

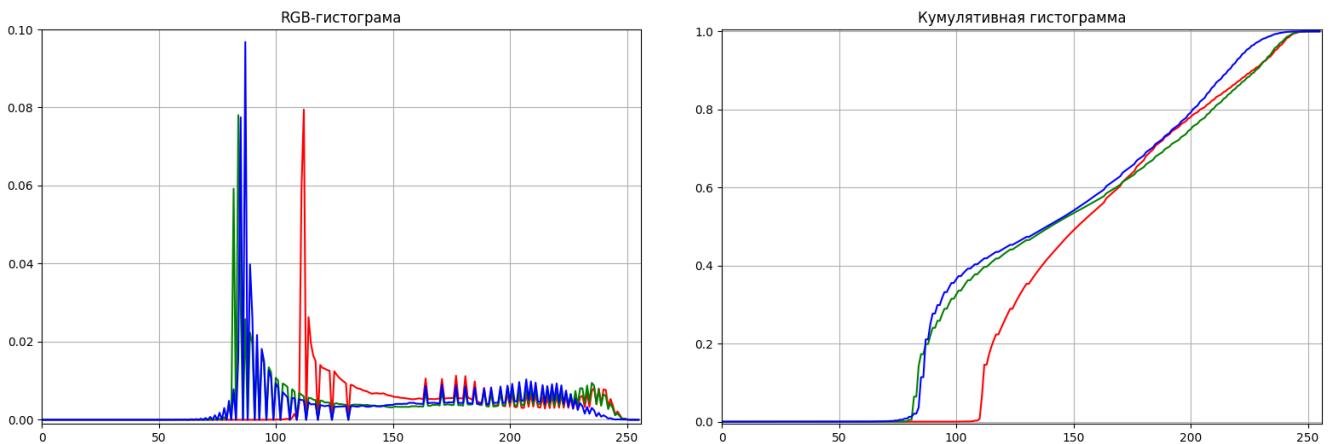
Таблица поиска — это метод для преобразования интенсивности пикселей изображения с помощью таблицы преобразования. В данном случае мы применим таблицу преобразования, которая будет преобразовывать интенсивность пикселей изображения по степенному закону с коэффициентом $\alpha = 0.5$. Рассмотрим код этого преобразования:

```

1 def LUT_conversion(image: MatLike, alpha: float) -> NDArray[np.uint8]:
2     """Преобразование изображения с помощью таблицы преобразования"""
3     lut = np.arange(256, dtype=np.uint8)
4     lut = (lut - image.min()) / (image.max() - image.min())
5     lut = np.where(lut > 0, lut, 0)
6     lut = np.clip(255 * np.power(lut, alpha), 0, 255)
7     return cv2.LUT(image, lut)
8
9
10 apply_method(LUT_conversion, image, '9LUT', alpha=0.5)

```

Листинг 11: Код, применяющий таблицу поиска к изображению



По RGB-гистограмме видно, что на тёмных градациях произошло разрежение гистограммы, а на светлых — наоборот, интенсивность приподнялась, при этом вся гистограмма сдвинулась вправо, что сделало изображение в целом ярче. Изображение воспринимается легче и на нём видно больше деталей благодаря поднятию интенсивности светлых пикселей. Преобразование подойдёт для любых изображений, т.к. таблица поиска строится индивидуально для каждого изображения, но там, где плюсы, там и минусы — не стоит ожидать сверхъестественного улучшения визуального качества.

Профили и проекции

Профиль изображения вдоль горизонтальной линии по середине

Профилем изображения называют функцию интенсивности изображения, распределённой вдоль некоторой линии (представьте себе сканер, который сканирует полоску шириной в 1 пиксель). В данном случае мы рассмотрим профиль изображения вдоль горизонтальной линии по середине изображения. Вот код для отображения профиля изображения:

```

1 def render_image_profile(img: MatLike, name: str):
2     """Рендерит профиль яркости изображения"""
3     clear()
4     profile = img.shape[0] // 2, []
5
6     fig = plt.figure()
7     fig.set_size_inches(654 / fig.dpi, 744 / fig.dpi)
8
9     ax1 = plt.subplot2grid((2, 1), (0, 0))
10    ax1.set_title('Исходное изображение')
11    ax1.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB), aspect='auto')
12
13    ax2 = plt.subplot2grid((2, 1), (1, 0), sharex=ax1)
14    ax2.set_xlabel('Профиль яркости', fontsize='large')
15    ax2.plot(profile)
16    plt.setp(ax2.get_xticklabels(), visible=False)
17
18    plt.savefig(f'{name}_profile.png')
19
20
21 barcode_image = cv2.imread('barcode.png')
22 render_image_profile(barcode_image, 'barcode')
```

Листинг 12: Код, отображающий профиль изображения



Результат на лицо, профиль изображения помогает в распознавании штрихкодов и QR-кодов, анализе медицинских изображений и текстур или в измерении контраста. И мы отчётливо видим, как профиль резкими перепадами показывает тёмные участки изображения — по сути полосы штрихкода.

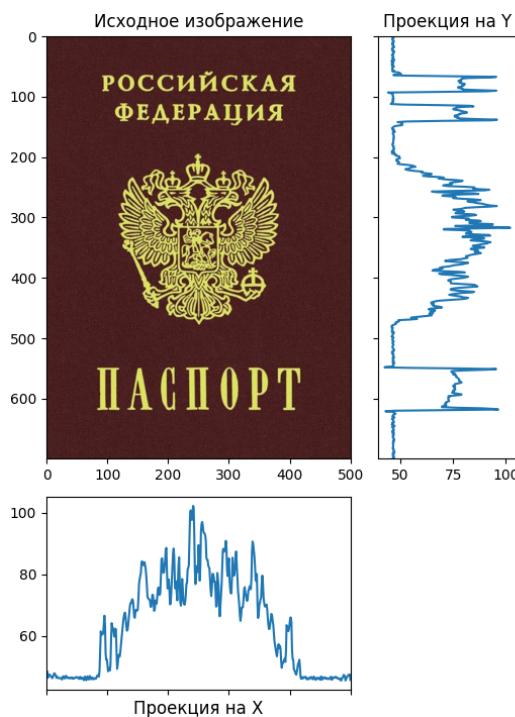
Проекция изображения на оси

Проекция изображения на ось — сумма интенсивности пикселей изображения в направлении, перпендикулярном этой оси (представьте, как прозрачный шарик катится по прямой линии и чем сильнее он будет окрашен в цвета изображения в конце пути, тем выше значение на графике). Рассмотрим проекции изображения на горизонтальную и вертикальную оси. Перед вами код для отображения этих проекций:

```

1 def render_projection(img: MatLike, name: str):
2     """Рендерит проекцию изображения"""
3     clear()
4     channels = cv2.split(img)
5     projection_x = sum(np.sum(channel, axis=0) for channel in channels) / img.shape[0] / 3
6     projection_y = sum(np.sum(channel, axis=1) for channel in channels) / img.shape[1] / 3
7
8     fig = plt.figure()
9     fig.set_size_inches(600 / fig.dpi, 840 / fig.dpi)
10
11    ax1 = plt.subplot2grid((3, 3), (0, 0), rowspan=2, colspan=2)
12    ax1.set_title('Исходное изображение')
13    ax1.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB), aspect='auto')
14
15    ax2 = plt.subplot2grid((3, 3), (2, 0), colspan=2, sharex=ax1)
16    ax2.set_xlabel('Проекция на X', fontsize='large')
17    ax2.plot(range(img.shape[1]), projection_x)
18    plt.setp(ax2.get_xticklabels(), visible=False)
19
20    ax3 = plt.subplot2grid((3, 3), (0, 2), rowspan=2, sharey=ax1)
21    ax3.set_title('Проекция на Y')
22    ax3.plot(projection_y, range(img.shape[0]))
23    plt.setp(ax3.get_yticklabels(), visible=False)
24
25    plt.savefig(f'{name}_projection.png')
26
27
28 passport_image = cv2.imread('passport.jpg')
29 render_projection(passport_image, 'passport')
```

Листинг 13: Код, отображающий проекции изображения на обе оси



Делаем вывод, что проекции полезны для анализа шаблонов и выделения отдельных объектов на монотонном изображении с получением их размеров и с последующим их обрезанием.

Вывод

В этой лабораторной работе мы освоили методы преобразования изображения для изменения его характеристик и улучшения визуального качества, чтобы в последующем упростить анализ изображения. Кроме того, мы изучили профили и проекции изображения, которые также помогают в анализе изображения и благодаря которым мы сможем выделять границы и объекты на изображении. К тому же мы познакомились с библиотекой `OpenCV` и библиотекой `Matplotlib`, которые упрощают работу с изображениями и визуализацией результатов — всё это помогло нам лучше понять, как работают представленные преобразования. В целом, лабораторная работа прошла успешно, и мы готовы двигаться дальше.

Напомним, что весь код из этой лабораторной работы и в том числе для генерации гистограмм и результатов можно найти в [приватном гисте в GitHub](#).

Вопросы к защите

1. Что такое контрастность изображения и как её можно изменить?

Контрастность изображения — это интервал между самым тёмным и самым ярким пикселим изображения. Её можно изменить с помощью различных преобразований, выбрав наиболее подходящие для конкретного изображения.

2. Чем эффективно использование профилей и проекций изображения?

Профили и проекции изображения помогают в определении границ и размеров объектов на монотонном изображении, а также в анализе шаблонов и обрезании изображений.

3. Каким образом можно найти объект на равномерном фоне?

Можно использовать проекции изображения на одну из осей, чтобы найти расположение объектов, а затем с помощью профилей определить форму и границы объекта на двумерной плоскости, чтобы выделить его.