Методические рекомендации к курсу «Введение в компьютерное зрение» для специальности 09.03.04 «Программная инженерия»

Никольская Ксения Юрьевна

2023

Цель дисциплины: Целью дисциплины является изучение и получение практических навыков для решения задач анализа видео- и графической информации. Задачи дисциплины: получение практических навыков работы с графической информацией; получение навыков создания наборов данных.

Задачи дисциплины: В рамках освоения дисциплины будут получены практические навыки по созданию наборов данных для обучения алгоритмов машинного обучения, по применению на практике различных функций специализированных библиотек для анализа изображений и видео (Pillow, OpenCV), развертыванию различных архитектур нейронных сетей для работы с видео- и графической информацией.

Основная литература по курсу:

1. Полупанов, Д.В. Программирование в Python 3: учебное пособие / Д.В. Полупанов, С.Р. Абдюшева, А.М. Ефимов. – Уфа: БашГУ, 2020. – 164 с. – ISBN 978-5-7477-5230-6. – Текст: электронный // Лань: электронно-библиотечная система. — Режим доступа: для авториз. пользователей.
2. Шакирьянов, Э.Д. Компьютерное зрение на Python. Первые шаги: учебное пособие / Э.Д. Шакирьянов. – Москва: Лаборатория знаний, 2021. – 163 с. – ISBN 978-5-00101-944-2. – Текст: электронный // Лань: электронно-библиотечная система. – Режим доступа: для авториз. пользователей.
3. Селянкин, В.В. Компьютерное зрение. Анализ и обработка изображений: учебник для вузов / В. В. Селянкин. – 2-е изд., стер. – Санкт-Петербург: Лань, 2021. – 152 с. – ISBN 978-5-8114-8259-7. – Текст: электронный // Лань: электронно-библиотечная система. – Режим доступа: для авториз. пользователей.
4. Алексеев, Д.С. Технологии интеллектуального анализа данных: учебное пособие / Д.С. Алексеев. – Кострома: КГУ им. Н.А. Некрасова, 2020. – 141 с. – ISBN 978-5-8285-1083-2. – Текст: электронный // Лань: электронно-библиотечная система. – Режим доступа: для авториз. пользователей.

Дополнительная литература по курсу:

1. Ян, Э. С. Программирование компьютерного зрения на языке Python / Э. С. Ян; перевод с английского А. А. Слинкин. – Москва: ДМК Пресс, 2016. – 312 с. – ISBN 978-5-97060-200-3. – Текст: электронный // Лань: электронно-библиотечная система. — Режим доступа: для авториз. пользователей.
2. Тоуманнен, Б. Программирование GPU при помощи Python и CUDA: руководство / Б. Тоуманнен; перевод с английского А. В. Борескова. – Москва: ДМК Пресс, 2020. – 252 с. – ISBN 978-5-97060-821-0. – Текст:

электронный // Лань: электронно-библиотечная система. – Режим доступа: для авториз. пользователей.

Объем и виды учебной работы:

- Семестр: 7.
- Общая трудоемкость дисциплины: 144 часа.
- Лекции: 16 часов.
- Практические занятия: 32 часа.

Компетенции:

1. ПК-9 (ПК-6 модели) Способен создавать и поддерживать системы искусственного интеллекта на основе нейросетевых моделей и методов:

Знает: ПК-6.1. З-2. Знает функциональность современных инструментальных средств и систем программирования в области создания моделей искусственных нейронных сетей.

Умеет: ПК-6.1. У-1. Умеет проводить оценку и выбор моделей искусственных нейронных сетей и инструментальных средств для решения задачи машинного обучения.

Имеет практический опыт: создания и обучения нейросетевых моделей для решения задач в области компьютерного зрения.

2. ПК-10 (ПК-9 модели) Способен создавать и внедрять одну или несколько сквозных цифровых субтехнологий искусственного интеллекта:

Знает: ПК-9.1. З-1. Знает принципы построения систем компьютерного зрения, методы и технологии искусственного интеллекта для анализа изображений и видео, методы и подходы к планированию и реализации проектов по созданию систем искусственного интеллекта на основе сквозной цифровой субтехнологии «Компьютерное зрение».

Умеет: ПК-9.1. У-1. Умеет применять методы и подходы к планированию и реализации проектов по созданию и поддержке системы искусственного интеллекта на основе сквозной цифровой субтехнологии «Компьютерное зрение».

Имеет практический опыт: создания и обучения модели искусственного интеллекта на основе сквозной цифровой субтехнологии «Компьютерное зрение».

Рекомендации к выполнению практических работ:

Рекомендации к выполнению **Практической работы 1** «OpenCV»:

Вопросы:

1. Из каких двух частей состоит имя файла?

2. Что такое OpenCV?

3. Какой метод может загрузить изображение из указанного файла?

4. Сколько возможных значений может принимать пиксель?

5. Куда будет указывать переменная, если не применять метод copy().

Рекомендации к выполнению **Практической работы 2** «Геометрические трансформации и специальные функции в библиотеке OpenCV»:

Вопросы:

1. С помощью какой функции из модуля cv2 можно изменить размер изображения?

2. Что оценивает интерполяция параметров?

3. Что такое translation?

4. Что можно изменить при помощи матрицы поворота?

5. Можно ли умножать элементы массивов одинаковой формы?

Рекомендации к выполнению **Практической работы 3** «Pillow Library (PIL)»:

Вопросы:

1. Если изображение отсутствует в рабочем каталоге, можно ли его загрузить, используя его полный путь?

2. Можно ли использовать атрибуты объекта изображения для получения информации?

3. Что такое RGB?

4. Какой метод загружает данные изображения в память компьютера?

5. Что такое квантование изображения?

Рекомендации к выполнению **Практической работы 4** «Геометрические трансформации и специальные функции в библиотеке Pillow»:

Вопросы:

1. Какие операции позволяют выполнить геометрические преобразования.

2. Какой метод позволяет повернуть изображение на нужный угол.

3. Сколько каналов имеет серое изображение.

4. Можно ли преобразовать изображение PIL в массив numpy.

5. Что используют пространственные операции для определения текущего значения пикселя.

Рекомендации к выполнению **Практической работы 5** «Основы работы в Keras»:

Вопросы:

1. С помощью какого метода можно добавить слои в Keras.
2. С помощью какого метода можно визуализировать модель в Keras.
3. Что такое параллелизм данных.
4. Что такое параллелизм устройств.
5. Что такое Сэмпл.

6. Как называется библиотека для публикации, обнаружения и использования повторно используемых частей моделей машинного обучения в TensorFlow.

7. Как называется фреймворк для машинного обучения и других вычислений на децентрализованных данных.

Рекомендации к выполнению **Практической работы 6** «Основы работы в PyTorch»:

Вопросы:

1. Верно ли утверждение, что PyTorch использует динамические вычислительные графы.
2. Верно ли утверждение, что динамические вычислительные графы не требуют компиляции перед каждым его выполнением.
3. С помощью чего происходит оптимизация производительности в PyTorch.
4. Перечислите плюсы и минусы PyTorch.

5. Что такое тензор в Python-библиотеке NumPy?

Рекомендации к выполнению **Практической работы 7** «Классификация изображений с использованием PyTorch»:

Вопросы:

1. Верно ли утверждение, что PyTorch использует динамические вычислительные графы.
2. Верно ли утверждение, что динамические вычислительные графы не требуют компиляции перед каждым его выполнением.
3. С помощью чего происходит оптимизация производительности в PyTorch.
4. Перечислите плюсы и минусы PyTorch.
5. Что такое тензор в Python-библиотеке NumPy?

Рекомендации к выполнению **Практической работы 8** «Сверточные нейронные сети»:

Вопросы:

1. Какие три архитектурные идеи объединяют сверточные нейронные сети.
2. Назовите 3 этапа влияющие на выбор топологии сверточной нейронной сети.
3. Верно ли утверждение, что если размер входных данных будет слишком велик, то вычислительная сложность повысится.
4. Верно ли утверждение, что если размер входных данных будет слишком маленький, то нейронная сеть не сможет выявить ключевые признаки.
5. Перечислите достоинства функции активации ReLU.
6. Перечислите недостатки функции активации ReLU.
7. Перечислите достоинства функции активации tahn.
8. Перечислите недостатки функции активации tahn.
9. Дайте определения ядра свертки.
10. Что дает подвыборочный слой.

Рекомендации к выполнению **Практической работы 9** «Рекуррентные нейронные сети:

Вопросы:

1. Дайте определение долгой краткосрочной памяти.

2. Верно ли утверждение, что LSTM разработаны специально, чтобы избежать проблемы долговременной зависимости.

3. Назовите ключевой компонент LSTM.

4. Дайте определение слоя фильтра забывания (forget gate layer).

5. Верно ли утверждение, что фильтр забывания контролирует убывание значения, хранящегося в памяти, т.е. до тех пор, пока входной и выходной фильтр не работают, значение градиента не меняется.

6. Процесс удаления информации из состояния ячейки осуществляется с помощью…

7. Слой фильтра утраты (вентиля забывания) отвечает за решение о том, …

8. Классический вариант LSTM впервые был предложен в …

9. Верно ли утверждение, что в сетях GRU вентиль обновления определяет, какую новую информацию нужно добавить к состоянию ячейки, а вентиль сброса – какую информацию сети нужно забыть.

10. Верно ли утверждение, что GRU имеет меньшее число параметров по сравнению с LSTM, что позволяет обучать сети GRU немного быстрее.

Вопросы к экзамену:

1. Из каких двух частей состоит имя файла?
2. Что такое OpenCV?
3. Какой метод может загрузить изображение из указанного файла?
4. Сколько возможных значений может принимать пиксель?
5. Куда будет указывать переменная, если не применять метод copy().
6. С помощью какой функции из модуля cv2 можно изменить размер изображения?
7. Что оценивает интерполяция параметров?
8. Что такое translation?
9. Что можно изменить при помощи матрицы поворота?
10. Можно ли умножать элементы массивов одинаковой формы?
11. Если изображение отсутствует в рабочем каталоге, можно ли его загрузить, используя его полный путь?
12. Можно ли использовать атрибуты объекта изображения для получения информации?
13. Что такое RGB?
14. Что такое квантование изображения?
15. Какие операции позволяют выполнить геометрические преобразования.
16. Какой метод позволяет повернуть изображение на нужный угол.
17. Сколько каналов имеет серое изображение.
18. Можно ли преобразовать изображение PIL в массив numpy.
19. Что используют пространственные операции для определения текущего значения пикселя.
20. С помощью какого метода можно добавить слои в Keras.
21. С помощью какого метода можно визуализировать модель в Keras.
22. Что такое параллелизм данных.
23. Что такое параллелизм устройств.
24. Что такое Сэмпл.
25. Как называется библиотека для публикации, обнаружения и использования повторно используемых частей моделей машинного обучения в TensorFlow.
26. Как называется фреймворк для машинного обучения и других вычислений на децентрализованных данных.
27. Верно ли утверждение, что PyTorch использует динамические вычислительные графы.
28. Верно ли утверждение, что динамические вычислительные графы не требуют компиляции перед каждым его выполнением.
29. С помощью чего происходит оптимизация производительности в PyTorch.
30. Перечислите плюсы и минусы PyTorch.

31. Что такое тензор в Python-библиотеке NumPy?
32. Какие три архитектурные идеи объединяют сверточные нейронные сети.
33. Назовите 3 этапа влияющие на выбор топологии сверточной нейронной сети.
34. Верно ли утверждение, что если размер входных данных будет слишком велик, то вычислительная сложность повысится.
35. Верно ли утверждение, что если размер входных данных будет слишком маленький, то нейронная сеть не сможет выявить ключевые признаки.
36. Перечислите достоинства функции активации ReLU.
37. Перечислите недостатки функции активации ReLU.
38. Перечислите достоинства функции активации tahn.
39. Перечислите недостатки функции активации tahn.
40. Дайте определения ядра свертки.
41. Что дает подвыборочный слой.
42. Дайте определение долгой краткосрочной памяти.
43. Верно ли утверждение, что LSTM разработаны специально, чтобы избежать проблемы долговременной зависимости.
44. Назовите ключевой компонент LSTM.
45. Дайте определение слоя фильтра забывания (forget gate layer).
46. Верно ли утверждение, что фильтр забывания контролирует убывание значения, хранящегося в памяти, т.е. до тех пор, пока входной и выходной фильтр не работают, значение градиента не меняется.
47. Процесс удаления информации из состояния ячейки осуществляется с помощью…
48. Слой фильтра утраты (вентиля забывания) отвечает за решение о том, …
49. Классический вариант LSTM впервые был предложен в …
50. Верно ли утверждение, что в сетях GRU вентиль обновления определяет, какую новую информацию нужно добавить к состоянию ячейки, а вентиль сброса – какую информацию сети нужно забыть.
51. Верно ли утверждение, что GRU имеет меньшее число параметров по сравнению с LSTM, что позволяет обучать сети GRU немного быстрее.

Практическая работа 1 "OpenCV"

1. Задание: Необходимо повторить весь код из примера с одним своим изображением, что означает, что все манипуляции должны быть произведены с одним изображением.
2. Прикрепить файл в формате .ipynb.
3. Балл за задание: 3.

Введение

Задачи обработки изображений и компьютерного зрения включают отображение, обрезку, переворачивание, поворот, сегментацию изображения, классификацию, восстановление изображения, распознавание изображений, создание изображений. Кроме того, работа с изображениями через облако требует хранения, передачи и сбора изображений через Интернет. Python - отличный выбор, поскольку в нем есть множество инструментов для обработки изображений, компьютерного зрения и библиотек искусственного интеллекта. Наконец, в нем есть множество библиотек для работы с файлами в облаке и в Интернете. Цифровое изображение - это просто файл на вашем компьютере. В этой практическрй работе вы получите представление об этих файлах и научитесь работать с этими файлами с некоторыми популярными библиотеками.

Загрузим любое изображение в формате .png:

Ввод [51]:

```
from IPython.display import Image
Image("Hogwarts.png")
```

Out[51]:

Ввод [52]:

```
def get_concat_h(im1, im2):
    #https://note.nkmk.me/en/python-pillow-concat-images/
    dst = Image.new('RGB', (im1.width + im2.width, im1.height))
    dst.paste(im1, (0, 0))
    dst.paste(im2, (im1.width, 0))
    return dst
```

Image Files and Paths

Ввод [53]:

```
my_image = "Hogwarts.png"
```

Имя файла состоит из двух частей: имени файла и расширения, разделенных точкой (`.`). Расширение определяет формат изображения. Существует два популярных формата изображений - изображение Объединенной группы экспертов по фотографии (или .jpg, .jpeg») и Portable Network Graphics (или .png). Эти типы файлов упрощают работу с изображениями. Например, он сжимает изображение с использованием аппроксимации синуса/косинуса, занимая меньше места на вашем диске для хранения изображения. Файлы изображений хранятся в файловой системе вашего компьютера. Его местоположение указывается с помощью «пути», который часто бывает уникальным. Вы можете найти путь к вашему текущему рабочему каталогу с помощью модуля Python `os` . Модуль `os` предоставляет функции для взаимодействия с файловой системой, например создание или удаление каталога (папки), перечисление ее содержимого, изменение и идентификация текущего рабочего каталога.

Ввод [54]:

```
import os
cwd = os.getcwd()
cwd
```

Out[54]:

```
'C:\\Users\\ppblondy\\Dropbox\\Универ лекции\\2022-2023\\1 семестр\\Поиск
и обработка\\Практическая_работа_№_2_Базовые_манипуляции_в_библиотеке_Open
CV'
```

«Путь» к изображению можно найти с помощью следующей строки кода.

Ввод [55]:

```
image_path = os.path.join(cwd, my_image)
image_path
```

Out[55]:

```
'C:\\Users\\ppblondy\\Dropbox\\Универ лекции\\2022-2023\\1 семестр\\Поиск
и обработка\\Практическая_работа_№_2_Базовые_манипуляции_в_библиотеке_Open
CV\\Hogwarts.png'
```

Загрузка изображений в Python

OpenCV - это библиотека, используемая для компьютерного зрения. У нее больше функциональных возможностей, чем у библиотеки `PIL` , но ее сложнее использовать. Мы можем импортировать OpenCV следующим образом:

Ввод [56]:

```
import cv2
```

Метод imread () загружает изображение из указанного файла, входными данными является путь к изображению, которое нужно прочитать (как и PIL), параметр флага указывает, как изображение должно быть прочитано, а значение по умолчанию - cv2.IMREAD_COLOR.

Ввод [57]:

```
image = cv2.imread(my_image)
```

Результатом является массив с большим количеством значений интенсивности в виде 8-битных целых чисел без знака.

Ввод [58]:

```
type(image)
```

Out[58]:

```
numpy.ndarray
```

Мы можем получить форму массива из атрибута shape.

Ввод [59]:

```
image.shape
```

Out[59]:

```
(1080, 1920, 3)
```

Форма такая же, как у массива PIL, но есть несколько отличий; например, PIL возвращается в формате (R, G, B), тогда как OpenCV возвращается в формате (B, G, R).

Каждый пиксель может принимать 256 возможных значений интенсивности в диапазоне от 0 до 255, где 0 - самая низкая интенсивность, а 255 - самая высокая. Максимальные и минимальные значения интенсивности изображения можно получить, соответственно, путем вызова:

Ввод [60]:

```
image.max()
```

Out[60]:

```
255
```

Ввод [61]:

```
image.min()
```

Out[61]:

```
0
```

Построение изображения

Вы можете использовать функцию OpenCV `imshow`, чтобы открыть изображение в новом окне, но это может вызвать некоторые проблемы в Jupyter:

Ввод [ ]:

```
cv2.imshow('image', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

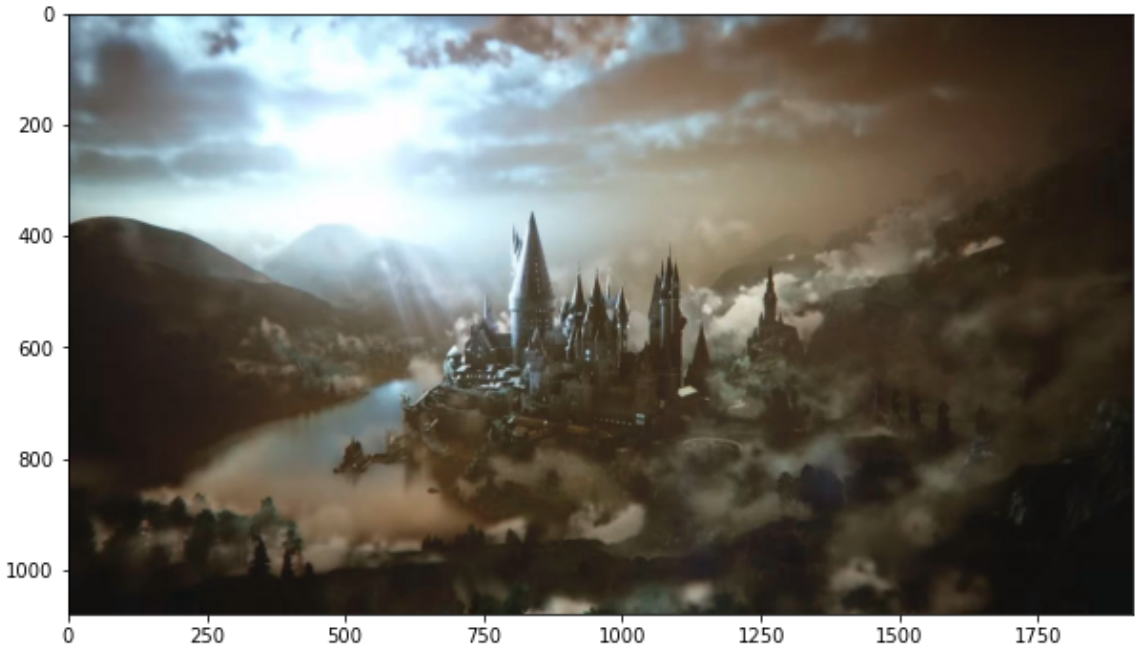
Вы также можете использовать функцию `imshow` из библиотеки `matplotlib`:

Ввод [62]:

```
import matplotlib.pyplot as plt
```

Ввод [63]:

```
plt.figure(figsize=(10,10))
plt.imshow(image)
plt.show()
```



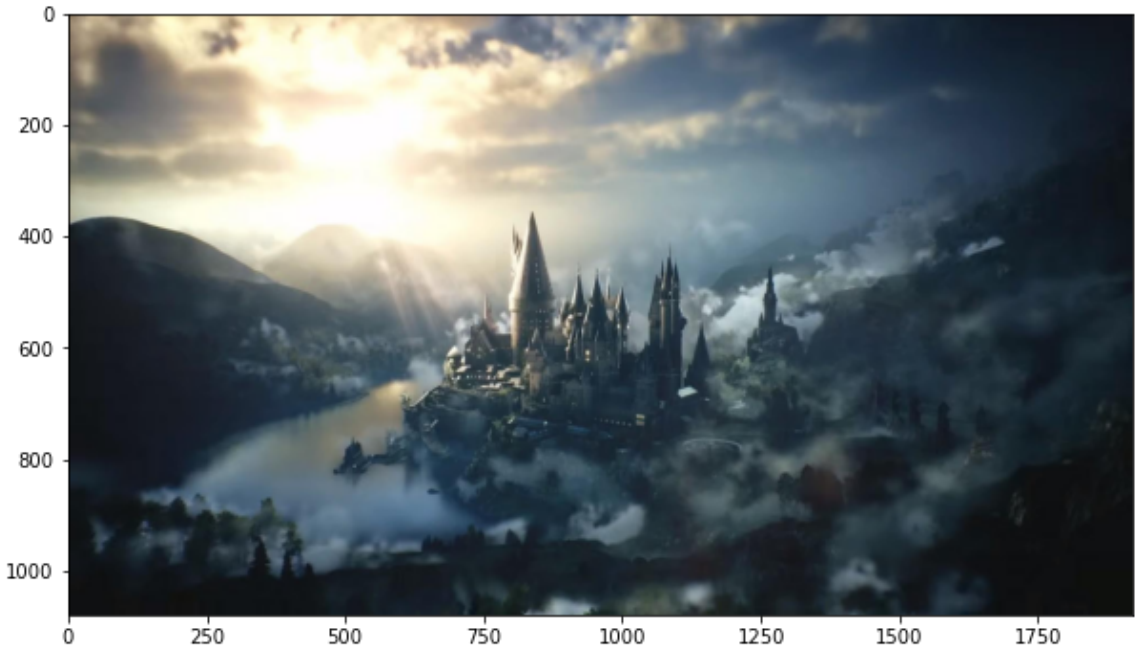
Изображение на выходе выглядит неестественным. Это потому, что порядок каналов RGB отличается. Мы можем изменить цветовое пространство с помощью кода преобразования и функции `cvtColor` из библиотеки `cv2`:

Ввод [64]:

```
new_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

Ввод [65]:

```
plt.figure(figsize = (10,10))
plt.imshow(new_image)
plt.show()
```



Вы также можете загрузить изображение, используя его путь, это пригодится, если изображение отсутствует в вашем рабочем каталоге:

Ввод [66]:

```
image_path
```

Out[66]:

```
'C:\\Users\\ppblondy\\Dropbox\\Универ лекции\\2022-2023\\1 семестр\\Поиск
и обработка\\Практическая_работа_№_2_Базовые_манипуляции_в_библиотеке_Open
CV\\Hogwarts.png'
```

Ввод [67]:

```
cv2.imwrite("Hogwarts.png", image)
```

Out[67]:

```
True
```

Изображения в оттенках серого

Изображения в градациях серого имеют значения пикселей, представляющие количество света или интенсивность. Светлые оттенки серого имеют высокую интенсивность, более темные оттенки имеют меньшую интенсивность. Белый цвет имеет самую высокую интенсивность, а черный - самую низкую.

Мы можем преобразовать изображение в оттенки серого, используя код преобразования цвета и функцию cvtColor.

Код для перехода от RGB к серому - cv2.COLOR_BGR2GRAY, мы применяем функцию:

Ввод [68]:

```
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Массив изображений имеет только два измерения, то есть только один цветовой канал:

Ввод [69]:

```
image_gray.shape
```

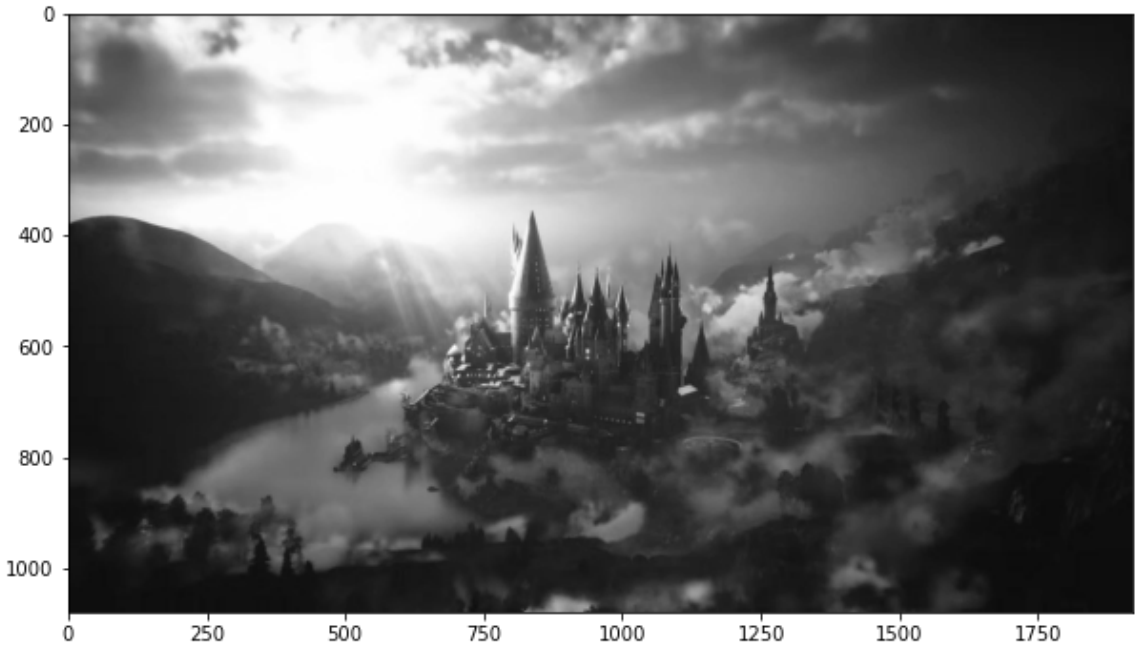
Out[69]:

```
(1080, 1920)
```

Мы можем построить изображение с помощью `imshow` , но мы должны указать, что цветовая карта серая:

Ввод [70]:

```
plt.figure(figsize=(10, 10))
plt.imshow(image_gray, cmap='gray')
plt.show()
```



Мы можем сохранить изображение как изображение в оттенках серого, давайте также сохраним его как `jpg` в рабочем каталоге.

Ввод [71]:

```
cv2.imwrite('Hogwarts1.jpg', image_gray)
```

Out[71]:

```
True
```

Вы также можете загрузить изображение в градациях серого, мы должны установить параметр флага на серый цвет кода диалога: cv2.COLOR_BGR2GRAY:

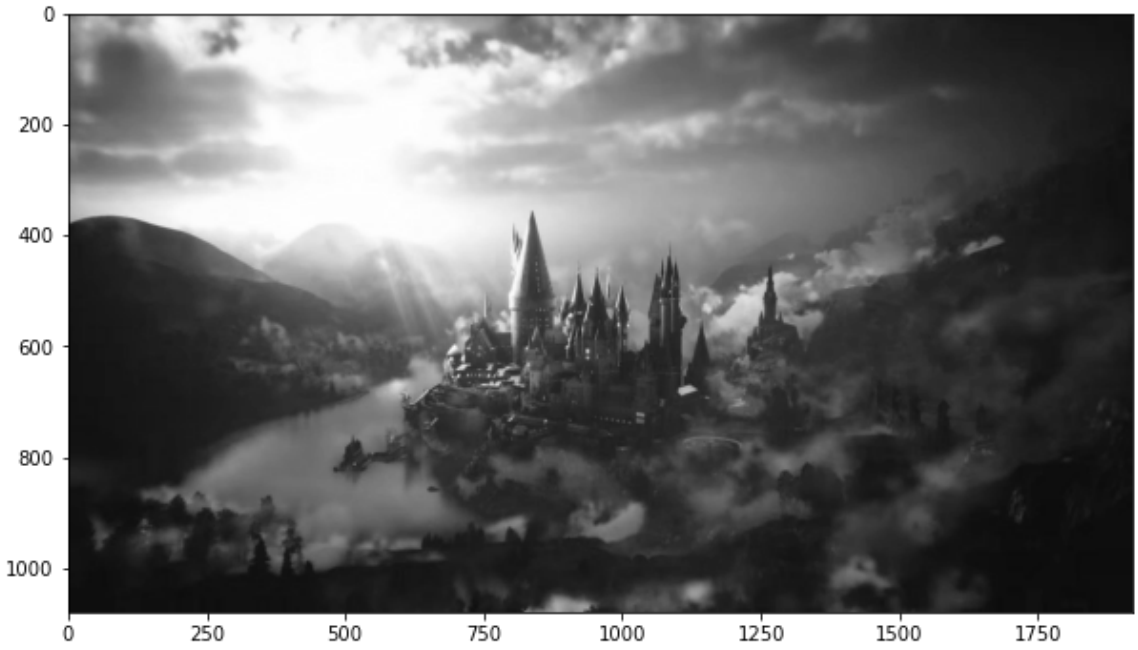
Ввод [72]:

```
im_gray = cv2.imread('Hogwarts.png', cv2.IMREAD_GRAYSCALE)
```

Мы можем построить изображение:

Ввод [73]:

```
plt.figure(figsize=(10,10))
plt.imshow(im_gray,cmap='gray')
plt.show()
```

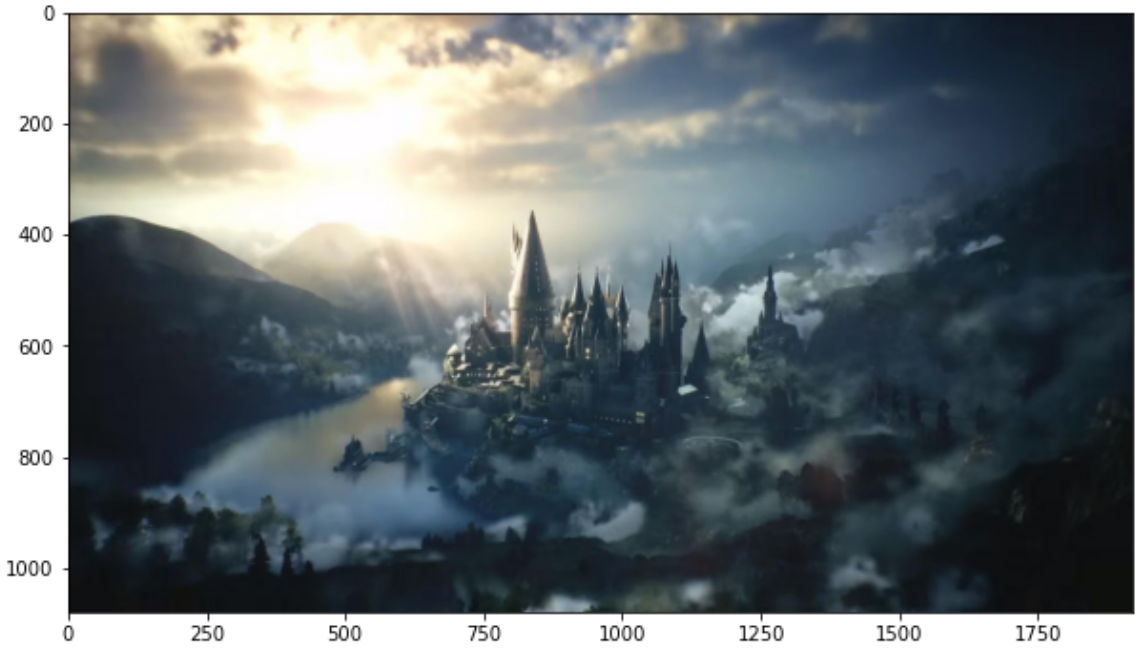


Цветовые каналы

Мы можем работать с разными цветовыми каналами. Рассмотрим следующее изображение:

Ввод [74]:

```
Hogwarts = cv2.imread('Hogwarts.png')
plt.figure(figsize = (10,10))
plt.imshow(cv2.cvtColor(Hogwarts, cv2.COLOR_BGR2RGB))
plt.show()
```



Мы можем получить разные цвета RGB и назначить их переменным синий, зеленый и красный в формате (B, G, R).

Ввод [75]:

```
blue, green, red = Hogwarts[:, :, 0], Hogwarts[:, :, 1], Hogwarts[:, :, 2]
```

Мы можем объединить каждый канал изображения с изображениями с помощью функции vconcat.

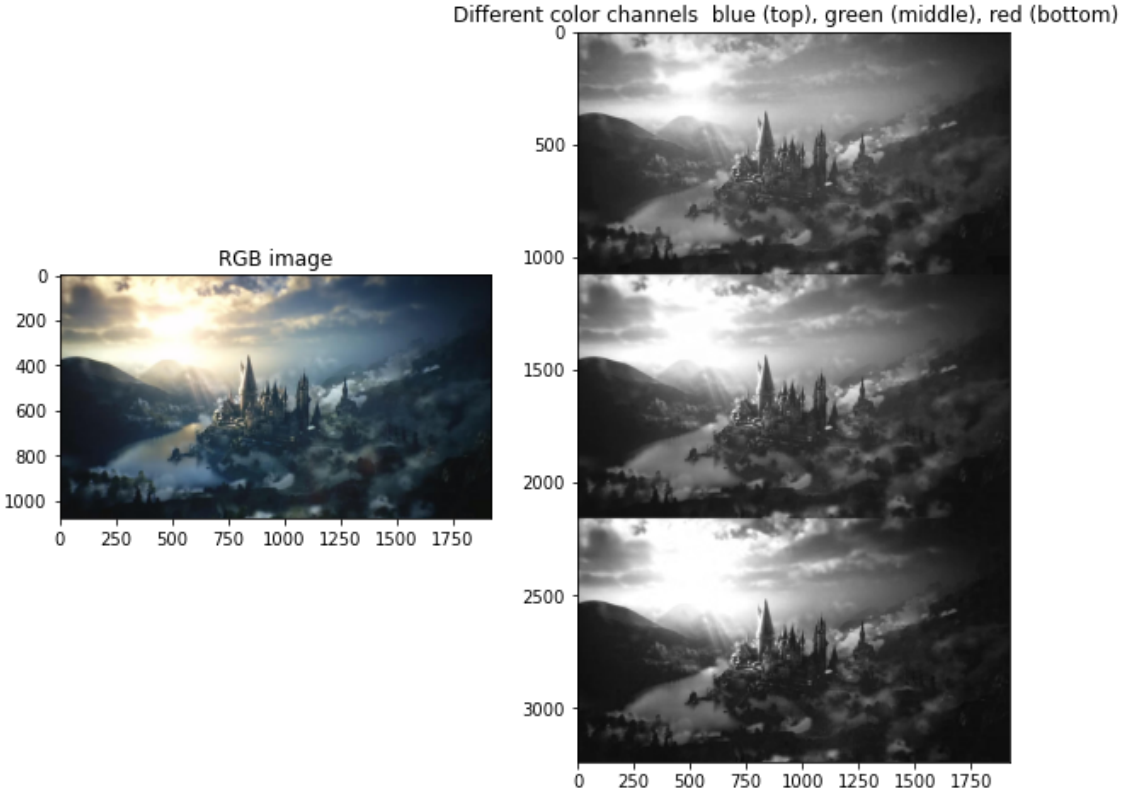
Ввод [76]:

```
im_bgr = cv2.vconcat([blue, green, red])
```

Нанося цветное изображение рядом с красным каналом в оттенках серого, мы видим, что области с красным цветом имеют более высокие значения интенсивности.

Ввод [77]:

```
plt.figure(figsize = (10,10))
plt.subplot(121)
plt.imshow(cv2.cvtColor(Hogwarts, cv2.COLOR_BGR2RGB))
plt.title("RGB image")
plt.subplot(122)
plt.imshow(im_bgr,cmap='gray')
plt.title("Different color channels  blue (top), green (middle), red (bottom)  ")
plt.show()
```



Indexing

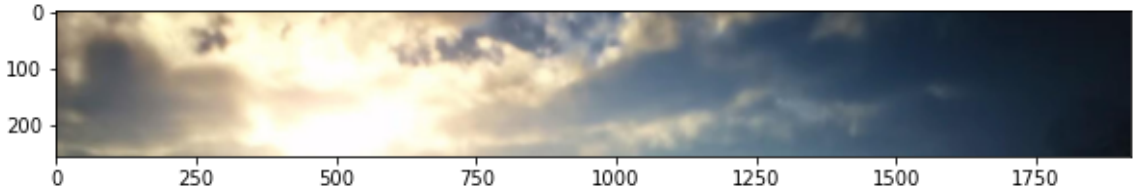
Мы можем использовать numpy нарезку. Например, мы можем вернуть первые 256 строк, соответствующих верхней половине изображения:

Ввод [78]:

```
rows = 256
```

Ввод [79]:

```
plt.figure(figsize=(10,10))
plt.imshow(new_image[0:rows,:,:])
plt.show()
```



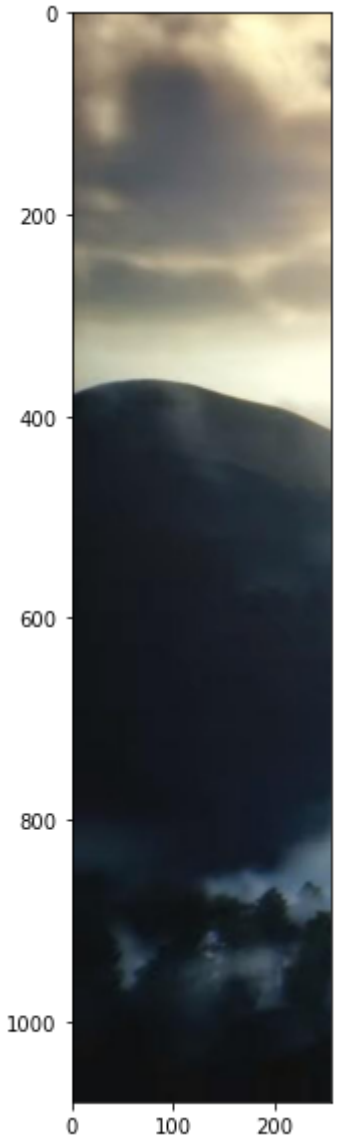
Мы можем вернуть первые 256 столбцов, соответствующих первой половине изображения:

Ввод [80]:

```
columns = 256
```

Ввод [81]:

```
plt.figure(figsize=(10,10))
plt.imshow(new_image[:,0:columns,:])
plt.show()
```



Если вы хотите переназначить массив другой переменной, вы должны использовать метод `copy` .

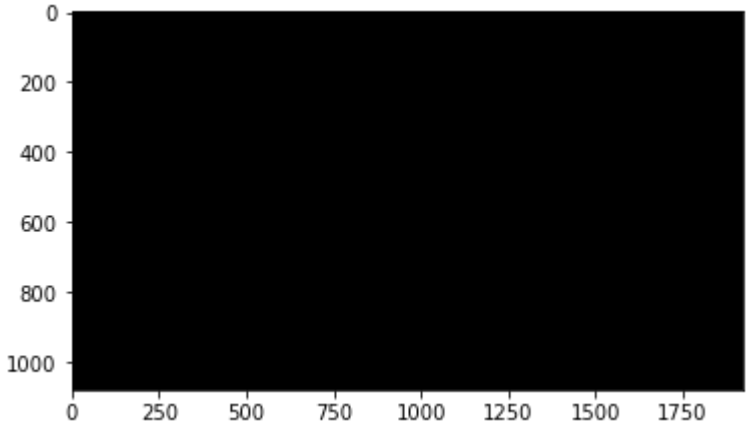
Ввод [82]:

```
A = new_image.copy()
plt.imshow(A)
plt.show()
```

Если мы не применим метод copy (), переменная будет указывать на то же место в памяти. Рассмотрим переменную `B` ниже, если мы установим все значения массива `A` равными нулю, поскольку `A` и `B` указывают на один и тот же объект в памяти, `B` также будет иметь нулевые элементы:

Ввод [83]:

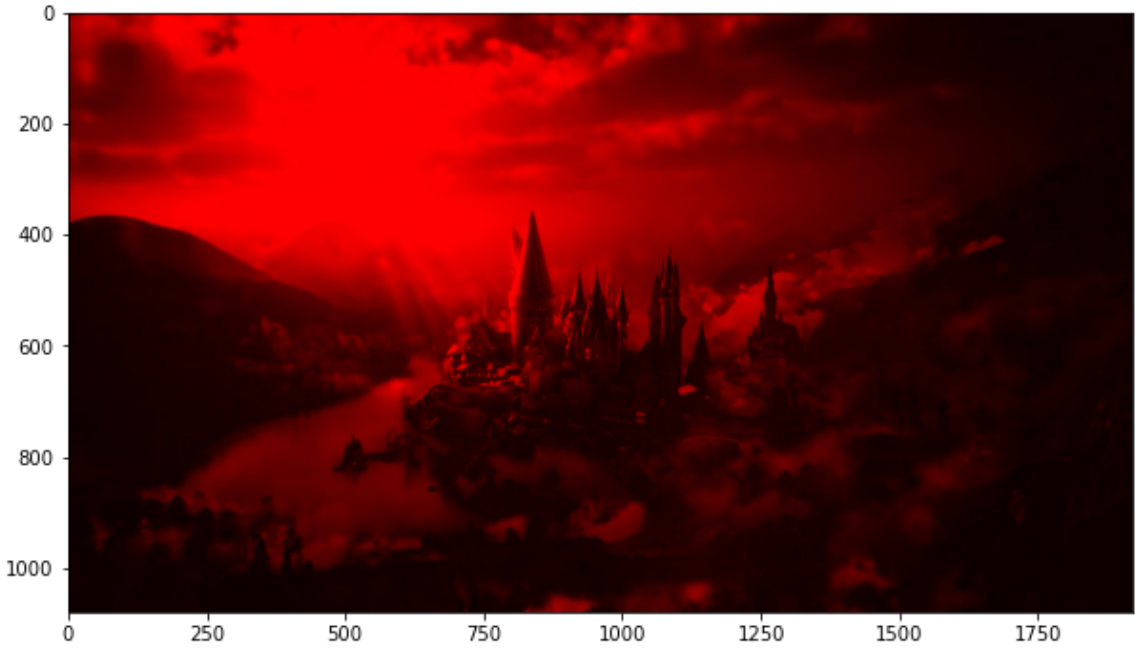
```
B = A
A[:,:,:] = 0
plt.imshow(B)
plt.show()
```



Мы также можем управлять элементами с помощью индексации. В следующем фрагменте кода мы создаем новый массив baboon_red и устанавливаем все каналы, кроме красного, равными нулю. Поэтому, когда мы показываем изображение, оно кажется красным:

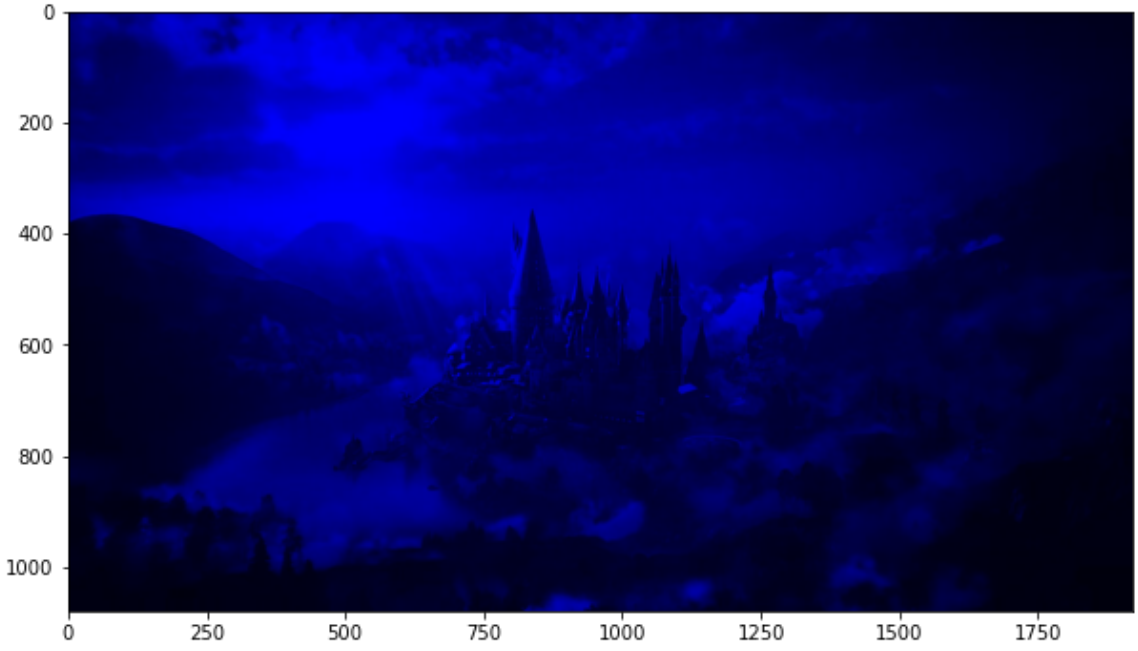
Ввод [84]:

```
Hogwarts_red = Hogwarts.copy()
Hogwarts_red[:, :, 0] = 0
Hogwarts_red[:, :, 1] = 0
plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(Hogwarts_red, cv2.COLOR_BGR2RGB))
plt.show()
```



Ввод [85]:

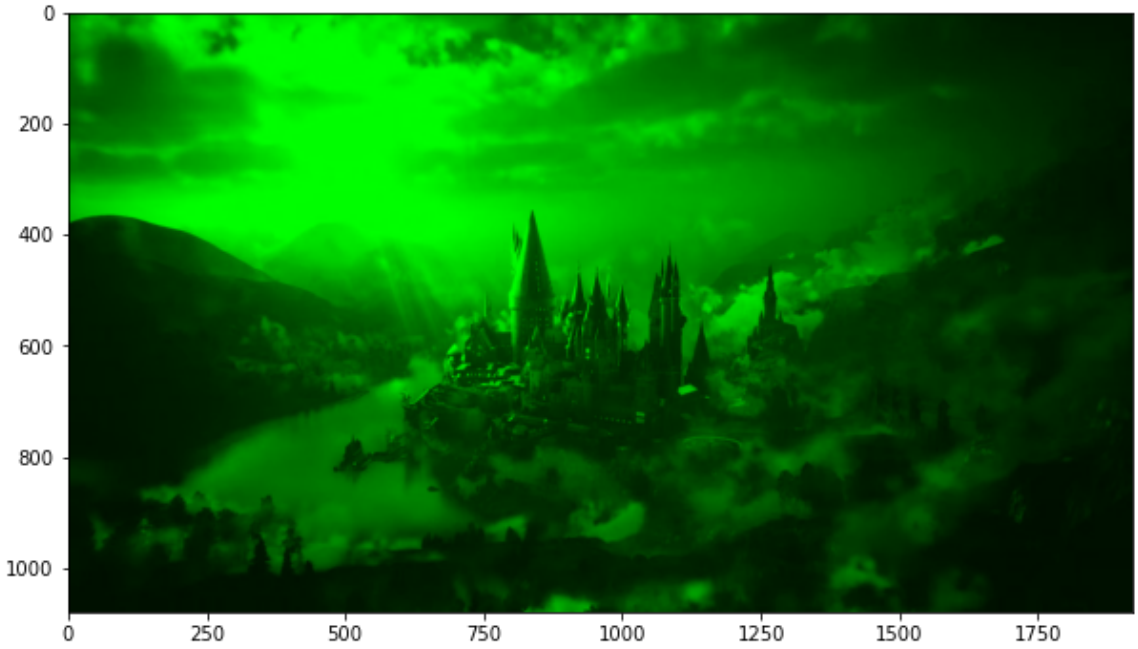
```
Hogwarts_blue = Hogwarts.copy()
Hogwarts_blue[:, :, 1] = 0
Hogwarts_blue[:, :, 2] = 0
plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(Hogwarts_blue, cv2.COLOR_BGR2RGB))
plt.show()
```



То же самое можно сделать и с зеленым:

Ввод [87]:

```
Hogwarts_green = Hogwarts.copy()
Hogwarts_green[:, :, 0] = 0
Hogwarts_green[:, :, 2] = 0
plt.figure(figsize=(10,10))
plt.imshow(cv2.cvtColor(Hogwarts_green, cv2.COLOR_BGR2RGB))
plt.show()
```

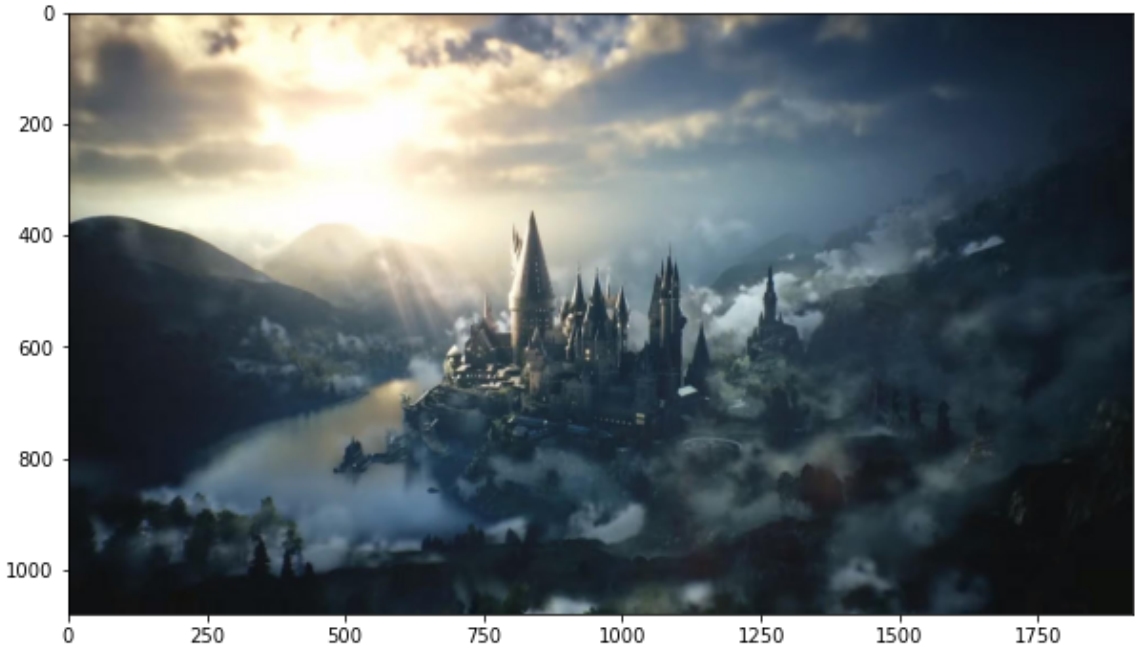


Ввод [88]:

```
image=cv2.imread('Hogwarts.png')
```

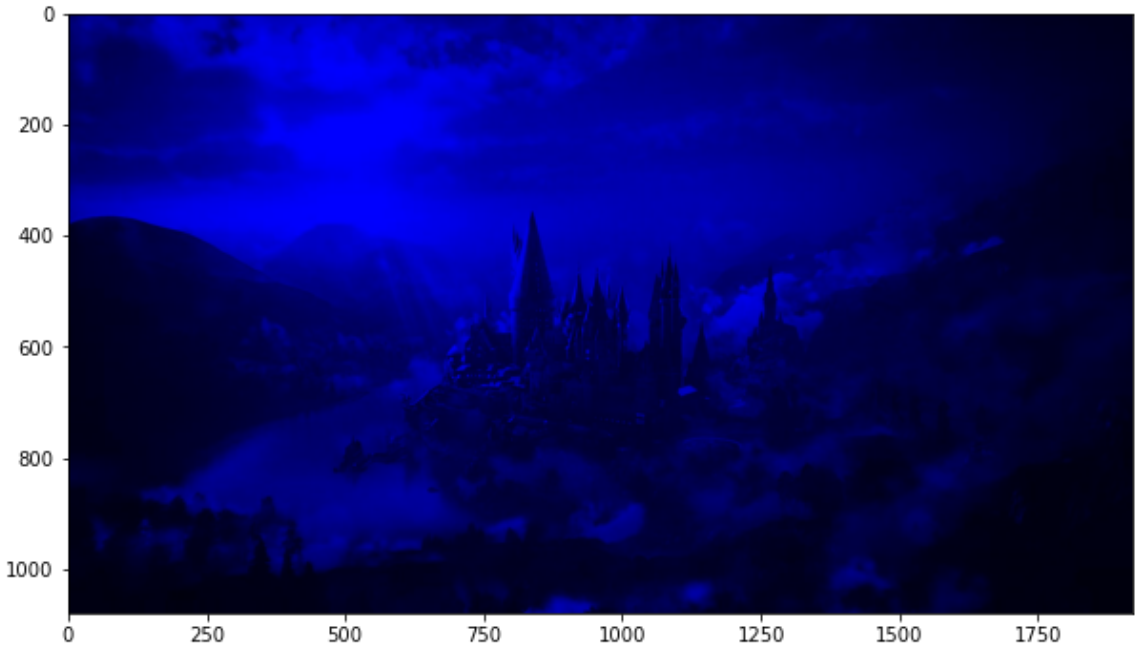
Ввод [89]:

```
plt.figure(figsize=(10,10))
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.show()
```



Ввод [91]:

```
image=cv2.imread('Hogwarts.png') # replace and add you image here name
Hogwarts_blue=image.copy()
Hogwarts_blue[:,:,1] = 0
Hogwarts_blue[:,:,2] = 0
plt.figure(figsize=(10,10))
plt.imshow(cv2.cvtColor(Hogwarts_blue, cv2.COLOR_BGR2RGB))
plt.show()
```



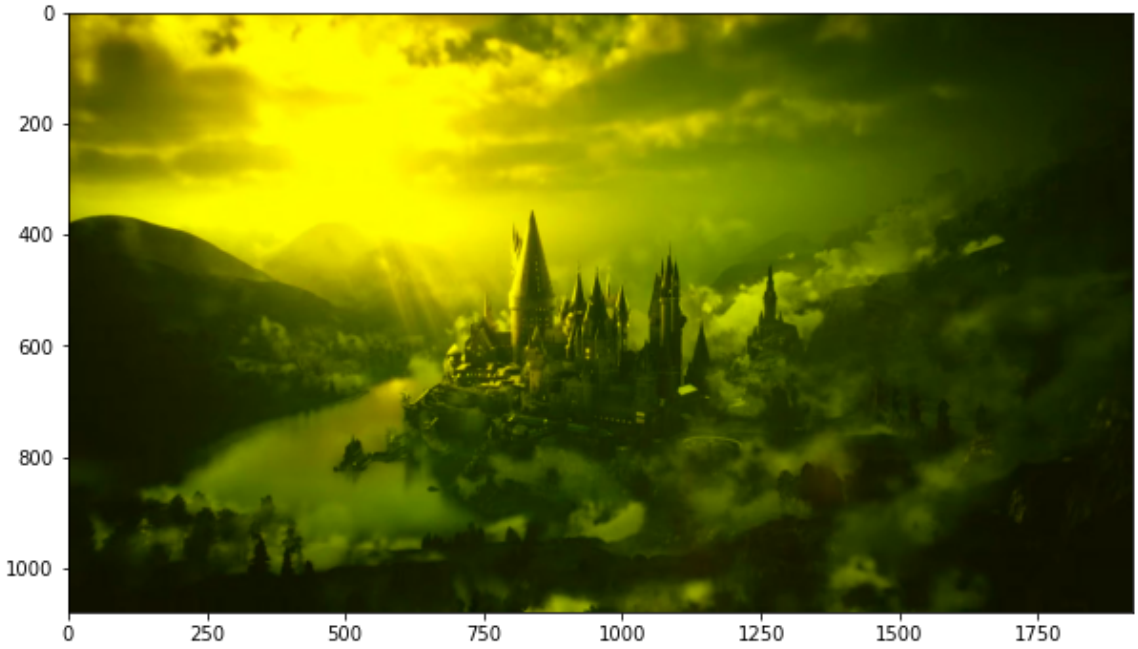
Откроем изображение и создадим объект изображения OpenCV с именем Hogwarts_blue, преобразуем изображение из формата BGR в формат RGB, извлекем из него синий канал и построим изображение.

Ввод [92]:

```
Hogwarts_blue[:,:,2] = 0
```

Ввод [93]:

```
Hogwarts_blue=cv2.imread('Hogwarts.png')
Hogwarts_blue=cv2.cvtColor(Hogwarts_blue, cv2.COLOR_BGR2RGB)
Hogwarts_blue[:,:,2] = 0
plt.figure(figsize=(10,10))
plt.imshow(Hogwarts_blue)
plt.show()
```



Ввод [ ]:

```

```

Практическая работа 2 "Геометрические трансформации и специальные функции в библиотеке OpenCV"

1. Задание: Необходимо повторить весь код из примера с одним своим изображением, что означает, что все манипуляции должны быть произведены с одним изображением.
2. Прикрепить файл в формате .ipynb.
3. Балл за задание: 3.

Введение

В этой практической работе вы будете применять геометрические преобразования к изображению. Это позволит вам выполнять различные операции, такие как изменение формы, сдвигать, изменять форму и поворачивать изображение. А также как применять к изображению некоторые базовые операции с массивами и матрицами.

Загрузим изображение:

Ввод [1]:

```
from IPython.display import Image
Image("1.png")
```

Out[1]:

Библиотечки:

Ввод [2]:

```
import matplotlib.pyplot as plt
import cv2
import numpy as np
```

Определим вспомогательную функцию для построения двух изображений рядом друг с другом.

Ввод [3]:

```
def plot_image(image_1, image_2,title_1="Orignal",title_2="New Image"):
    plt.figure(figsize=(10,10))
    plt.subplot(1, 2, 1)
    plt.imshow(image_1,cmap="gray")
    plt.title(title_1)
    plt.subplot(1, 2, 2)
    plt.imshow(image_2,cmap="gray")
    plt.title(title_2)
    plt.show()
```

Геометрические преобразования

Геометрические преобразования позволяют выполнять различные операции, например, сдвигать, изменять форму и поворачивать изображение.

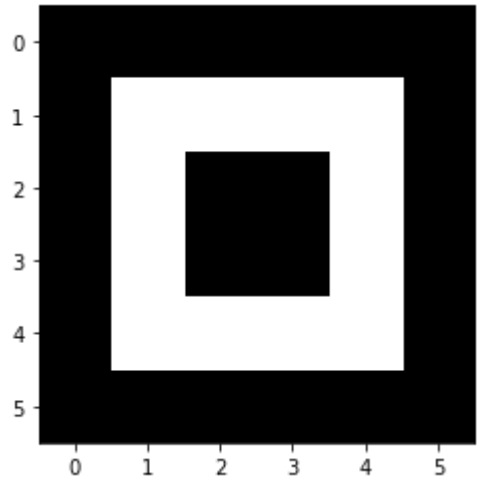
Масштабирование

Мы можем изменить размер изображения, используя для этой цели функцию `resize()` из модуля `cv2` . Вы можете указать коэффициент масштабирования или размер изображения:

Рассмотрим следующее изображение с соответствующими значениями интенсивности:

Ввод [4]:

```
toy_image = np.zeros((6,6))
toy_image[1:5,1:5]=255
toy_image[2:4,2:4]=0
plt.imshow(toy_image,cmap='gray')
plt.show()
toy_image
```



Out[4]:

```
array([[  0.,   0.,   0.,   0.,   0.,   0.],
       [  0., 255., 255., 255., 255.,   0.],
       [  0., 255.,   0.,   0., 255.,   0.],
       [  0., 255.,   0.,   0., 255.,   0.],
       [  0., 255., 255., 255., 255.,   0.],
       [  0.,   0.,   0.,   0.,   0.,   0.]])
```

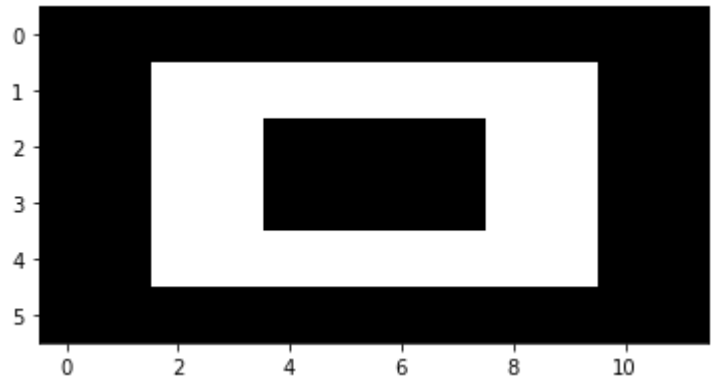
Мы можем изменить масштаб по определенной оси::

- `fx` : масштабный коэффициент по горизонтальной оси
- `fy` : масштабный коэффициент по вертикальной оси

Интерполяция параметров оценивает значения пикселей на основе соседних пикселей. `INTER_NEAREST` использует ближайший пиксель, а `INTER_CUBIC` использует несколько пикселей рядом со значением пикселя, которое мы хотели бы оценить.

Ввод [5]:

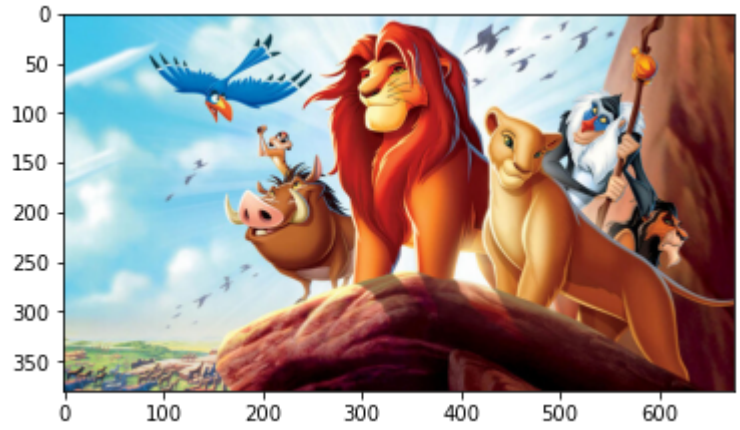
```
new_toy = cv2.resize(toy_image,None,fx=2, fy=1, interpolation = cv2.INTER_NEAREST )
plt.imshow(new_toy,cmap='gray')
plt.show()
```



Поэкспериментируем:

Ввод [6]:

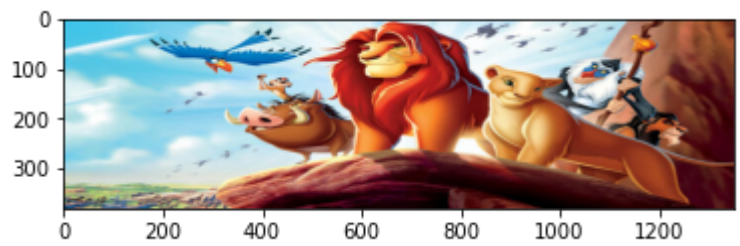
```
image = cv2.imread("1.png")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.show()
```



Мы можем масштабировать горизонтальную ось на два и оставить вертикальную ось как есть:

Ввод [7]:

```
new_image = cv2.resize(image, None, fx=2, fy=1, interpolation=cv2.INTER_CUBIC)
plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
plt.show()
print("old image shape:", image.shape, "new image shape:", new_image.shape)
```

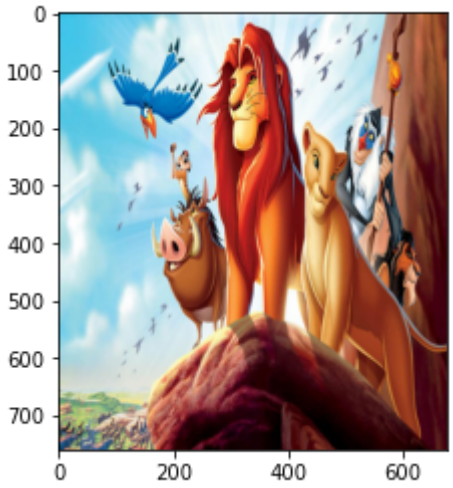



```
old image shape: (380, 676, 3) new image shape: (380, 1352, 3)
```

Таким же образом мы можем масштабировать вертикальную ось на два:

Ввод [8]:

```
new_image = cv2.resize(image, None, fx=1, fy=2, interpolation=cv2.INTER_CUBIC)
plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
plt.show()
print("old image shape:", image.shape, "new image shape:", new_image.shape)
```

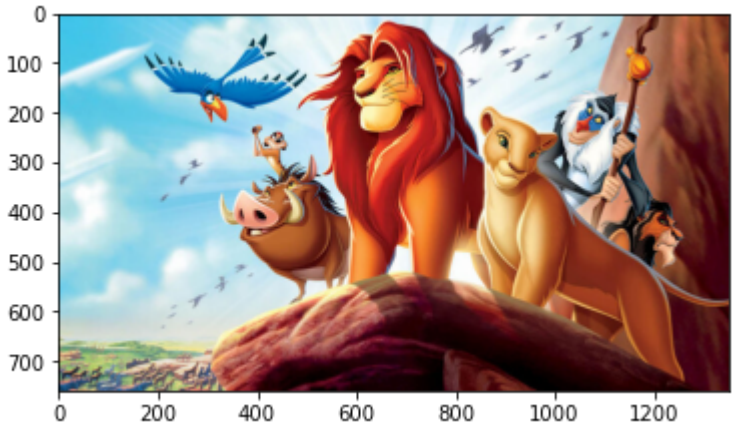



```
old image shape: (380, 676, 3) new image shape: (760, 676, 3)
```

Мы можем масштабировать горизонтальную ось и вертикальную ось на два.

Ввод [9]:

```
new_image = cv2.resize(image, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)
plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
plt.show()
print("old image shape:", image.shape, "new image shape:", new_image.shape)
```

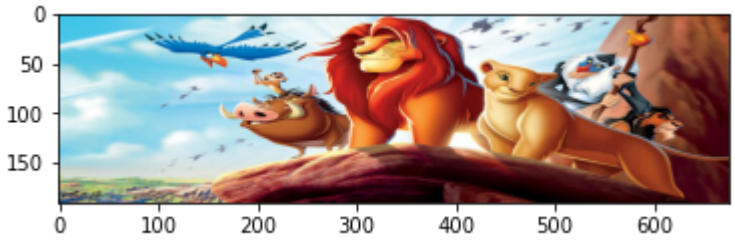



```
old image shape: (380, 676, 3) new image shape: (760, 1352, 3)
```

Мы также можем уменьшить изображение, установив коэффициент масштабирования на действительное число от 0 до 1:

Ввод [10]:

```
new_image = cv2.resize(image, None, fx=1, fy=0.5, interpolation=cv2.INTER_CUBIC)
plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
plt.show()
print("old image shape:", image.shape, "new image shape:", new_image.shape)
```




```
old image shape: (380, 676, 3) new image shape: (190, 676, 3)
```

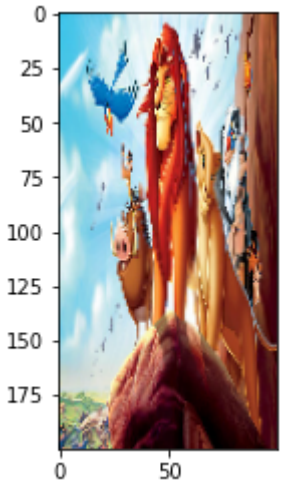
Мы также можем указать количество строк и столбцов:

Ввод [11]:

```
rows = 100
cols = 200
```

Ввод [12]:

```
new_image = cv2.resize(image, (100, 200), interpolation=cv2.INTER_CUBIC)
plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
plt.show()
print("old image shape:", image.shape, "new image shape:", new_image.shape)
```




```
old image shape: (380, 676, 3) new image shape: (200, 100, 3)
```

Translation (сдвиг/смещение)

Сдвиг/смещение — это когда вы меняете расположение изображения. `tx` — это количество пикселей, на которое вы смещаете местоположение в горизонтальном направлении, а `ty` — это количество пикселей, которое вы смещаете в вертикальном направлении. Вы можете создать матрицу преобразования M для сдвига изображения.

В этом примере мы сдвигаем изображение на 100 пикселей по горизонтали:

Ввод [13]:

```
tx = 100
ty = 0
M = np.float32([[1, 0, tx], [0, 1, ty]])
M
```

Out[13]:

```
array([[  1.,   0., 100.],
       [  0.,   1.,   0.]], dtype=float32)
```

Форма изображения определяется:

Ввод [14]:

```
rows, cols, _ = image.shape
```

Мы используем функцию `warpAffine` из модуля `cv2` . Первый входной параметр — массив изображений, второй входной параметр — матрица преобразования `M` , а последний входной параметр — длина и ширина выходного изображения $(cols, rows)$:

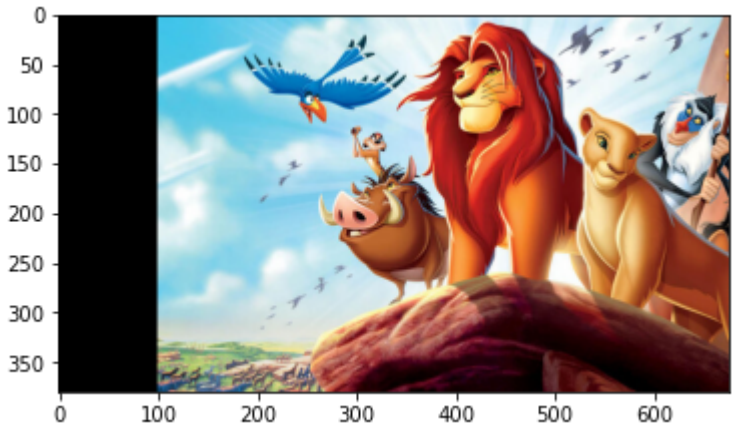
Ввод [15]:

```
new_image = cv2.warpAffine(image, M, (cols, rows))
```

Мы можем построить изображение; части изображения, которые не имеют никакой интенсивности, устанавливаются равными нулю:

Ввод [16]:

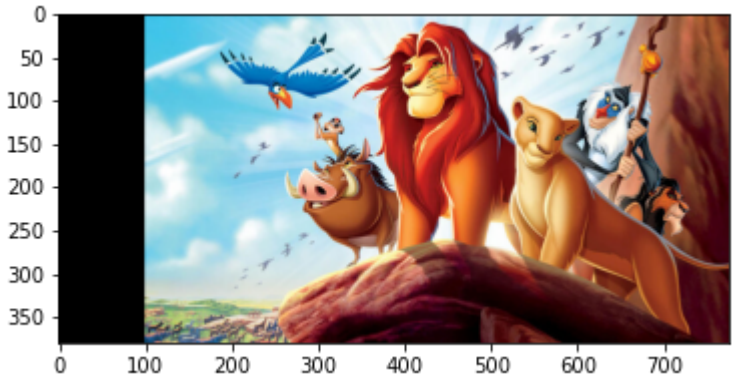
```
plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
plt.show()
```



Мы видим, что часть исходного изображения была обрезана. Мы можем исправить это, изменив размер выходного изображения: `(cols + tx,rows + ty)` :

Ввод [17]:

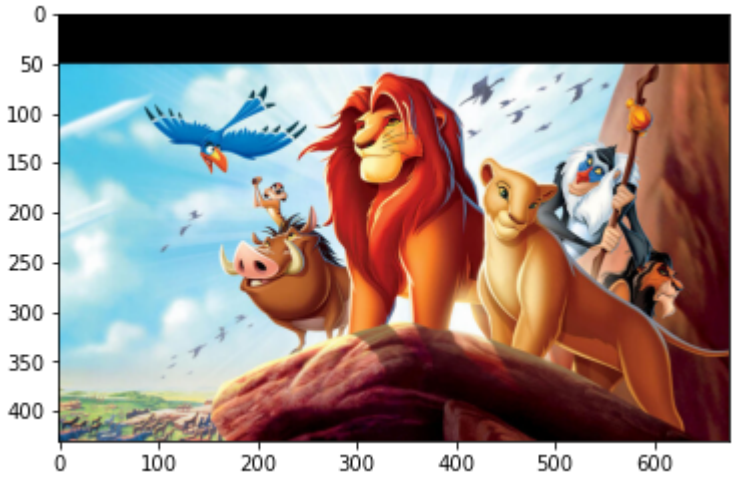
```
new_image = cv2.warpAffine(image, M, (cols + tx, rows + ty))
plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
plt.show()
```



Мы можем сдвинуть изображение по горизонтали:

Ввод [18]:

```
tx = 0
ty = 50
M = np.float32([[1, 0, tx], [0, 1, ty]])
new_iamge = cv2.warpAffine(image, M, (cols + tx, rows + ty))
plt.imshow(cv2.cvtColor(new_iamge, cv2.COLOR_BGR2RGB))
plt.show()
```



Вращение

Мы можем повернуть изображение на угол θ, что достигается матрицей вращения `getRotationMatrix2D` .

`center` : центр вращения исходного изображения. Мы будем использовать только центр изображения.

`угол` : угол поворота в градусах. Положительные значения означают вращение против часовой стрелки (предполагается, что начало координат находится в верхнем левом углу).

`масштаб` : Изотропный коэффициент масштабирования, в этом курсе значение будет равно единице.

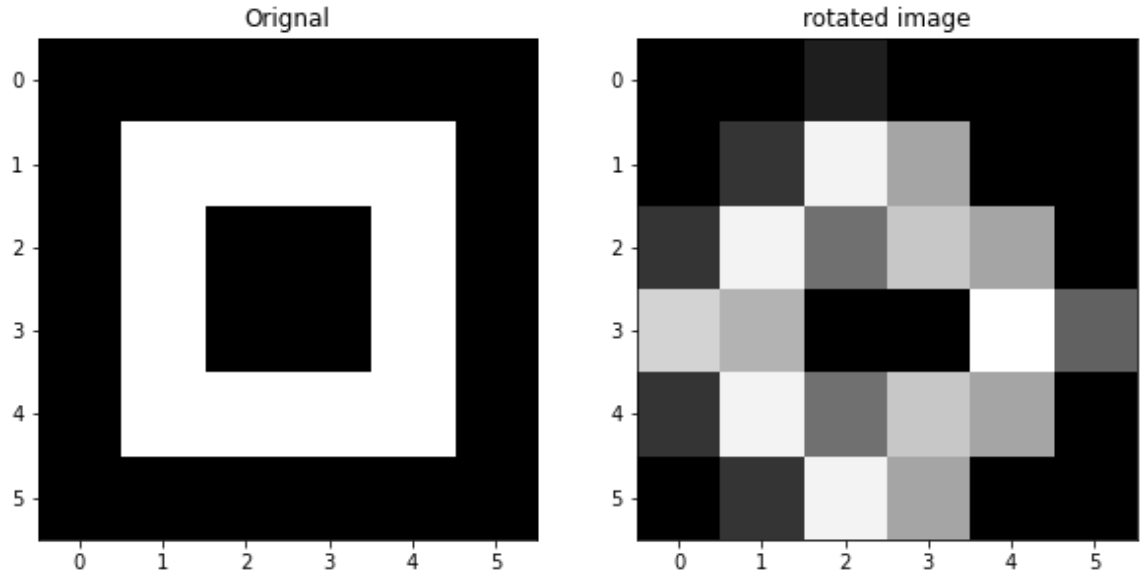
Мы можем повернуть наше изображение игрушки на 45 градусов:

Ввод [20]:

```
theta = 45.0
M = cv2.getRotationMatrix2D(center=(3, 3), angle=theta, scale=1)
new_toy_image = cv2.warpAffine(toy_image, M, (6, 6))
```

Ввод [21]:

```
plot_image(toy_image, new_toy_image, title_1="Orignal", title_2="rotated image")
```



Глядя на значения интенсивности, мы видим, что многие значения были интерполированы:

Ввод [22]:

```
new_toy_image
```

Out[22]:

```
array([[  0.        ,   0.        ,  28.38867188,   0.        ,
          0.        ,   0.        ],
       [  0.        ,  47.8125    , 223.125     , 151.40625   ,
          0.        ,   0.        ],
       [ 47.8125    , 223.125     , 103.59375   , 183.28125   ,
        151.40625   ,   0.        ],
       [195.234375  , 165.10253906,   0.        ,   0.        ,
        234.82910156,  89.89746094],
       [ 47.8125    , 223.125     , 103.59375   , 183.28125   ,
        151.40625   ,   0.        ],
       [  0.        ,  47.8125    , 223.125     , 151.40625   ,
          0.        ,   0.        ]])
```

Мы можем выполнить ту же операцию с цветными изображениями:

Ввод [23]:

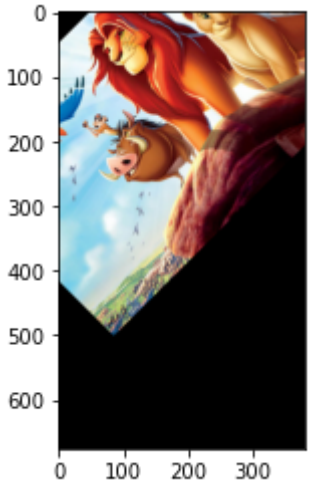
```
cols, rows, _ = image.shape
```

Ввод [24]:

```
M = cv2.getRotationMatrix2D(center=(cols // 2 - 1, rows // 2 - 1), angle=theta, scale=1)
new_image = cv2.warpAffine(image, M, (cols, rows))
```

Ввод [25]:

```
plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
plt.show()
```



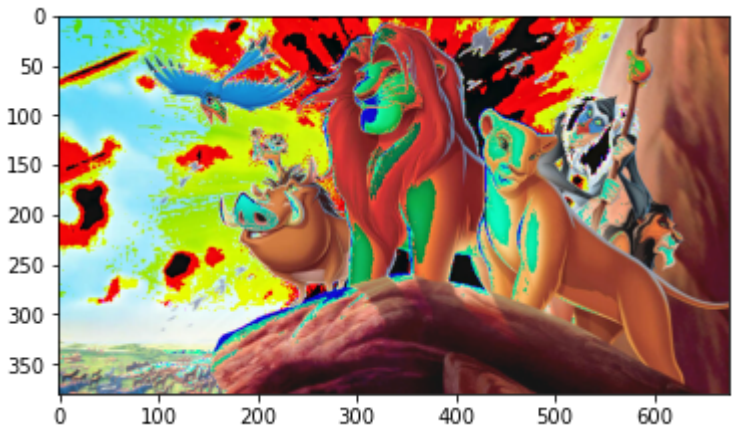
Математические операции

Операции с массивами

Мы можем выполнять операции с массивами над изображением, используя вещание, мы можем добавить константу к значению интенсивности каждого пикселя.

Ввод [26]:

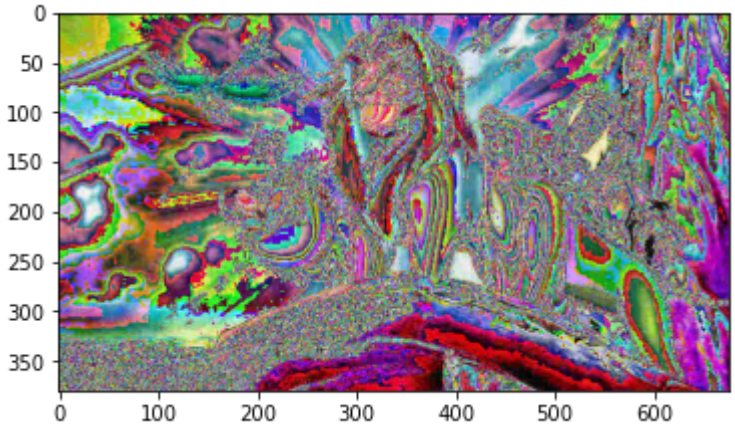
```
new_image = image + 20

plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
plt.show()
```



Мы также можем умножить значение интенсивности каждого пикселя на постоянное значение.

Ввод [27]:

```
new_image = 10 * image
plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
plt.show()
```



Мы можем сложить элементы двух массивов одинаковой формы. В этом примере мы генерируем массив случайных шумов той же формы и типа данных, что и наше изображение.

Ввод [28]:

```
Noise = np.random.normal(0, 20, (cols, rows,  3)).astype(np.uint8)
Noise.shape
```

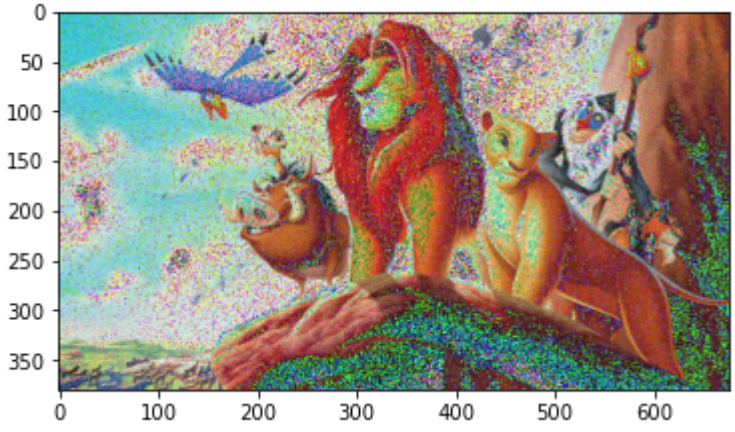
Out[28]:

```
(380, 676, 3)
```

Мы добавляем сгенерированный шум к изображению и строим результат. Мы видим значения, которые испортили изображение:

Ввод [29]:

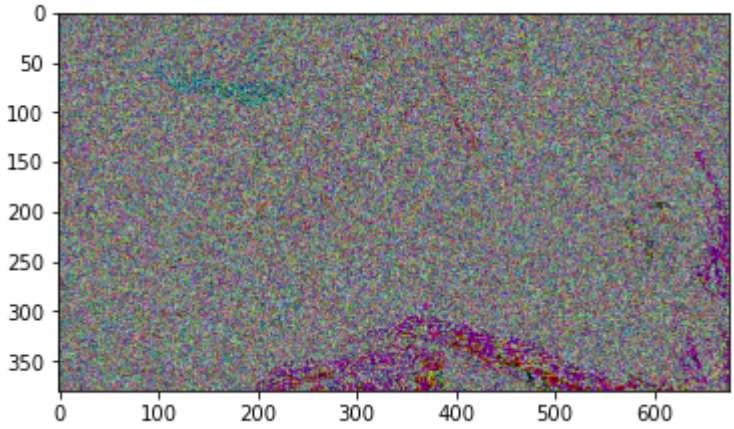
```
new_image = image + Noise

plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
plt.show()
```



При этом мы можем перемножать элементы двух массивов одинаковой формы. Мы можем умножить случайное изображение и 1 изображение и построить результат.

Ввод [30]:

```
new_image = image*Noise

plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
plt.show()
```

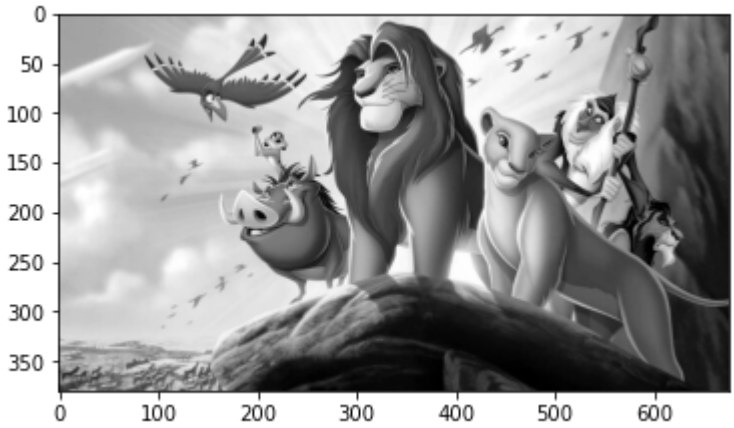


Матричные операции

Изображения в градациях серого являются матрицами. Рассмотрим следующее изображение в градациях серого:

Ввод [31]:

```
im_gray = cv2.imread('1.png', cv2.IMREAD_GRAYSCALE)
im_gray.shape

plt.imshow(im_gray,cmap='gray')
plt.show()
```



Мы можем применять алгоритмы, разработанные для матриц. Мы можем использовать разложение по сингулярным значениям, разлагая нашу матрицу изображения на произведение трех матриц.

Ввод [32]:

```
U, s, V = np.linalg.svd(im_gray , full_matrices=True)
```

Мы видим, что `s` не прямоугольный:

Ввод [33]:

```
s.shape
```

Out[33]:

```
(380,)
```

Мы можем преобразовать `s` в диагональную матрицу `S`:

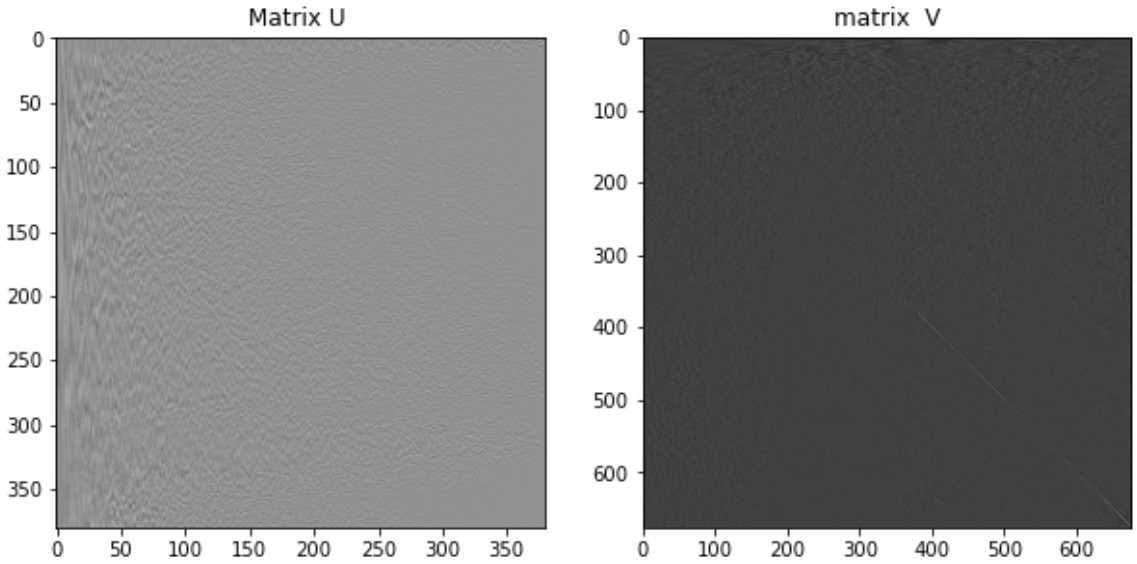
Ввод [34]:

```
S = np.zeros((im_gray.shape[0], im_gray.shape[1]))
S[:image.shape[0], :image.shape[0]] = np.diag(s)
```

Мы можем построить матрицу `U` и `V`:

Ввод [36]:

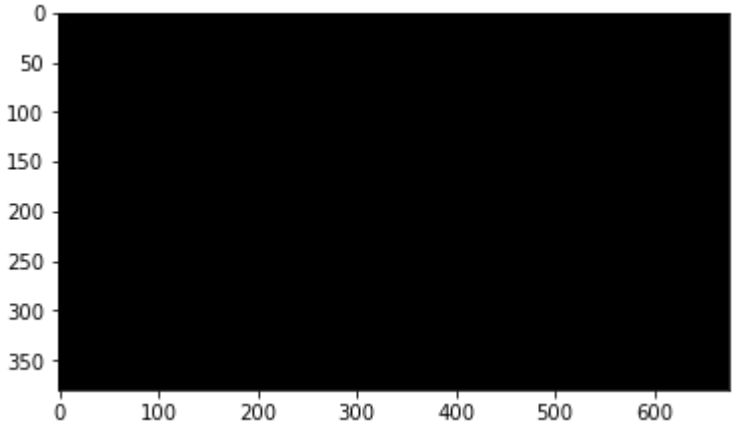
```
plot_image(U,V,title_1="Matrix U ",title_2="matrix  V")
```



Мы видим, что большинство элементов в `S` равны нулю:

Ввод [37]:

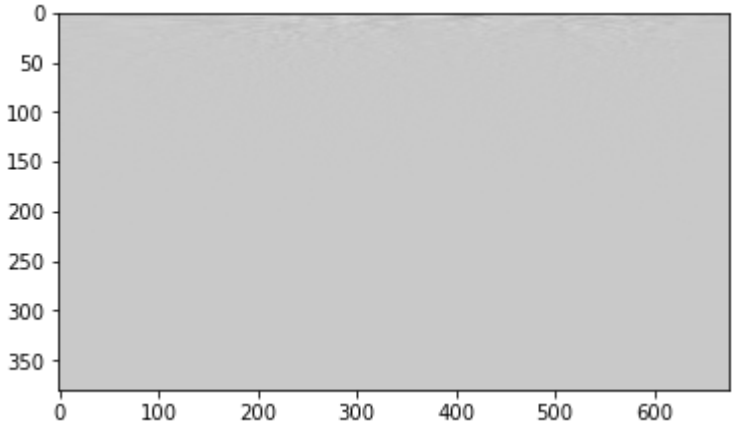
```
plt.imshow(S,cmap='gray')
plt.show()
```



Мы можем найти матричное произведение всех матриц. Во-первых, мы можем выполнить матричное умножение на `S` и `U` и присвоить его `B` и построить результаты:

Ввод [38]:

```
B = S.dot(V)
plt.imshow(B,cmap='gray')
plt.show()
```



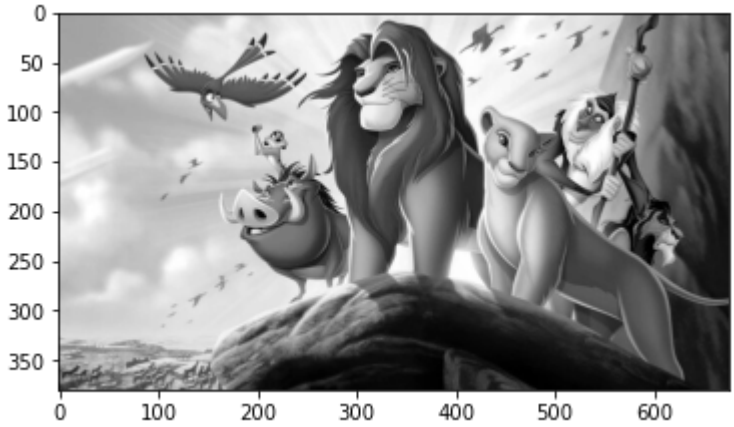
Мы можем найти матричное произведение `U`, `S` и `B`. Мы видим, что это все изображение:

Ввод [39]:

```
A = U.dot(B)
```

Ввод [40]:

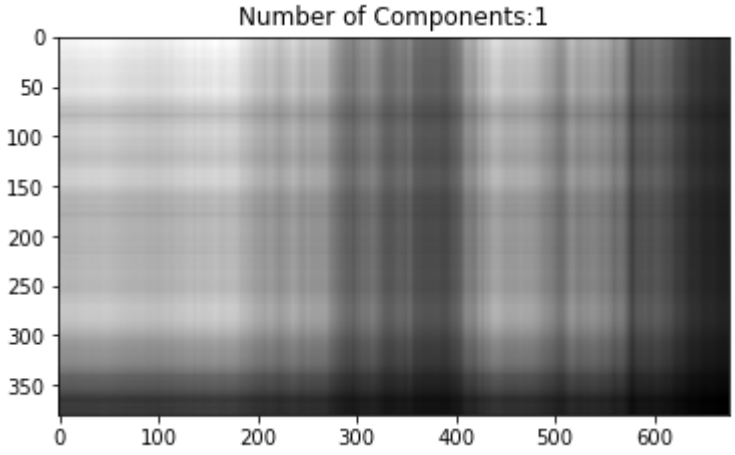
```
plt.imshow(A,cmap='gray')
plt.show()
```

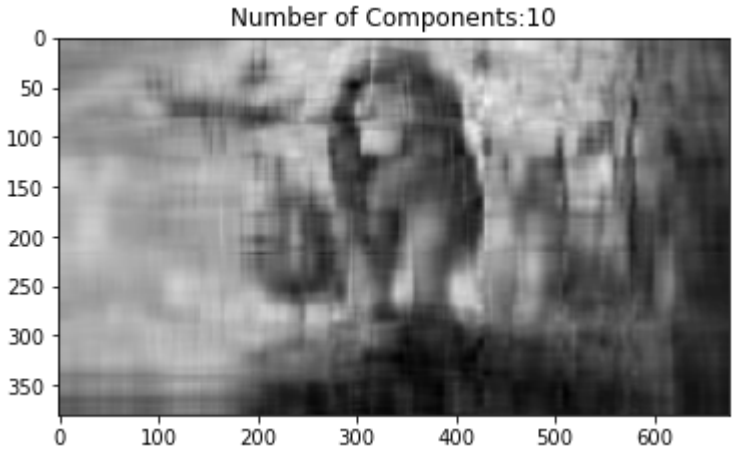


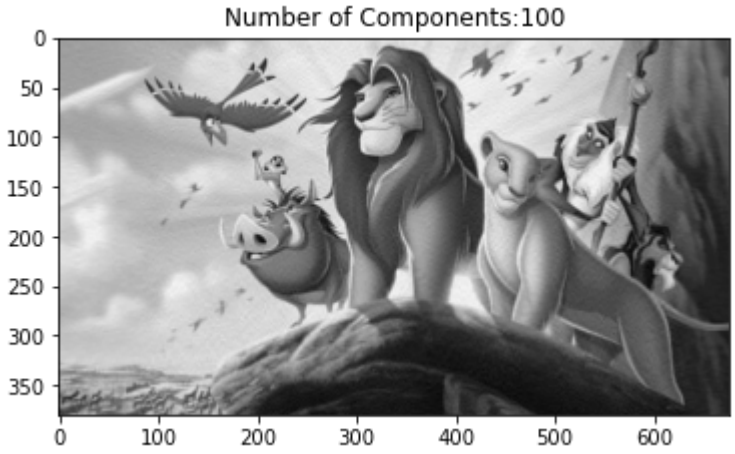
Оказывается, многие элементы избыточны, поэтому мы можем удалить некоторые строки и столбцы `S` и `V` и аппроксимировать изображение, найдя произведение.

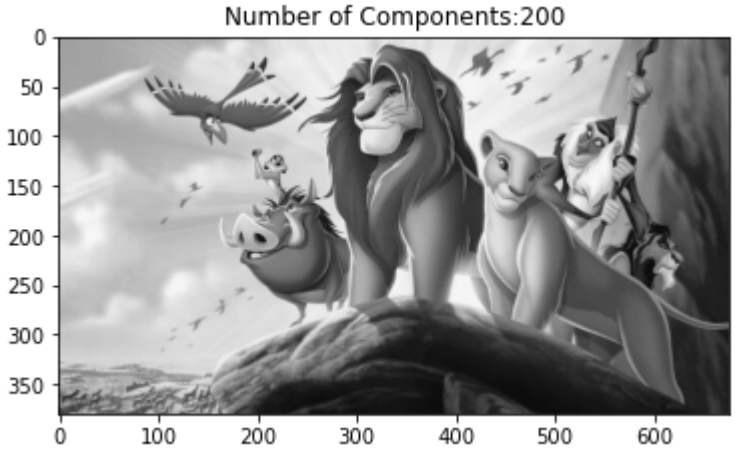
Ввод [41]:

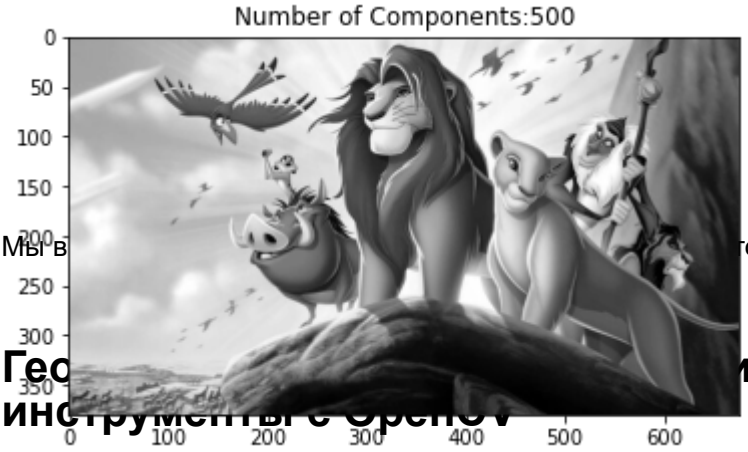
```
for n_component in [1,10,100,200, 500]:
    S_new = S[:, :n_component]
    V_new = V[:n_component, :]
    A = U.dot(S_new.dot(V_new))
    plt.imshow(A,cmap='gray')
    plt.title("Number of Components:"+str(n_component))
    plt.show()
```











Мы в тов для представления изображения.

Гео ие математические инструменты с OpenCV

Пространственные операции используют соседние пиксели для определения текущего значения пикселя. Приложения включают фильтрацию и повышение резкости. Они используются на многих этапах компьютерного зрения, таких как сегментация, и являются ключевым строительным блоком в алгоритмах искусственного интеллекта.

Загрузим изображение:

Ввод [42]:

```
from IPython.display import Image
Image("2.png")
```

Out[42]:

Библиотечки:

Ввод [43]:

```
# Used to view the images
import matplotlib.pyplot as plt
# Used to perform filtering on an image
import cv2
# Used to create kernels for filtering
import numpy as np
```

Функция для отображения двух изображения рядом:

Ввод [44]:

```
def plot_image(image_1, image_2,title_1="Orignal",title_2="New Image"):
    plt.figure(figsize=(10,10))
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(image_1, cv2.COLOR_BGR2RGB))
    plt.title(title_1)
    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(image_2, cv2.COLOR_BGR2RGB))
    plt.title(title_2)
    plt.show()
```

Линейная фильтрация

Фильтрация включает улучшение изображения, например, удаление шума из изображения. Шум вызван плохой камерой или плохим сжатием изображения. Те же самые факторы, которые вызывают шум, могут привести к размытию изображений. Мы можем применять фильтры для повышения резкости этих изображений. Свертка — это стандартный способ фильтрации изображения. Фильтр называется ядром, и разные ядра выполняют разные задачи. Свертка используется для многих самых передовых алгоритмов искусственного интеллекта. Мы просто берем скалярное произведение ядра и часть изображения одинакового размера. Затем мы сдвигаем ядро и повторяем.

Рассмотрим следующее изображение:

Ввод [45]:

```
# Loads the image from the specified file
image = cv2.imread("2.png")
print(image)
# Converts the order of the color from BGR (Blue Green Red) to RGB (Red Green Blue) then
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

```
[[[130 126 175]
  [132 126 173]
  [132 126 173]
  ...
  [ 78 104 116]
  [ 67  93 105]
  [ 53  79  91]]

 [[130 126 175]
  [133 127 174]
  [132 126 173]
  ...
  [ 53  81  92]
  [ 47  75  86]
  [ 41  69  80]]

 [[132 126 175]
  [132 126 173]
  [133 125 172]
  ...
  [ 27  57  68]
  [ 29  59  70]
  [ 25  55  66]]

 ...

 [[146 218 248]
  [146 216 246]
  [151 221 250]
  ...
  [ 34  18  41]
  [ 36  20  43]
  [ 40  21  46]]

 [[165 229 253]
  [162 227 248]
  [167 230 251]
  ...
  [ 31  15  38]
  [ 33  17  40]
  [ 38  20  43]]

 [[171 233 251]
  [171 230 249]
  [173 232 248]
  ...
  [ 32  17  38]
  [ 33  17  40]
  [ 40  22  45]]]
```

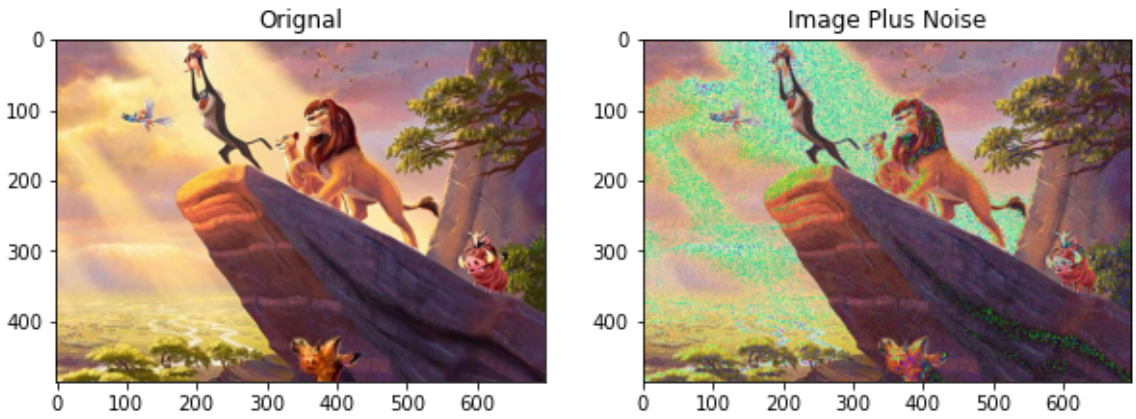
Out[45]:

```
<matplotlib.image.AxesImage at 0x1c336094850>
```

Изображения, с которыми мы работаем, состоят из значений RGB, которые представляют собой значения от 0 до 255. Ноль означает белый шум, из-за этого изображение выглядит зернистым:

Ввод [46]:

```
# Get the number of rows and columns in the image
rows, cols,_= image.shape
# Creates values using a normal distribution with a mean of 0 and standard deviation of 1
noise = np.random.normal(0,15,(rows,cols,3)).astype(np.uint8)
# Add the noise to the image
noisy_image = image + noise
# Plots the original image and the image with noise using the function defined at the top
plot_image(image, noisy_image, title_1="Orignal",title_2="Image Plus Noise")
```



При добавлении шума к изображению иногда значение может быть больше 255, в этом случае 256 вычитается из значения, чтобы обернуть число вокруг, сохраняя его между 0 и 255. Например, рассмотрим изображение со значением RGB 137. Мы добавляем шум со значением RGB 215 и получаем значение RGB 352. Затем мы вычитаем 256, общее количество возможных значений от 0 до 255, чтобы получить число от 0 до 255.

Фильтрация шума

Сглаживающие фильтры усредняют пиксели в окрестности, их иногда называют фильтрами нижних частот. Для средней фильтрации ядро просто усредняет ядра в окрестности.

Ввод [48]:

```
# Создаем ядро, представляющее собой массив 6 на 6, где каждое значение равно 1/36.
kernel = np.ones((6,6))/36
```

Функция `filter2D` выполняет 2D-свертку между изображением `src` и `kernel` для каждого цветового канала независимо. Параметр ddepth (https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkCV0101ENCoursera25797139-2021-01-01#filter_depths) имеет отношение к размеру выходного изображения, мы установим его равным -1, чтобы вход и выход имели одинаковый размер.

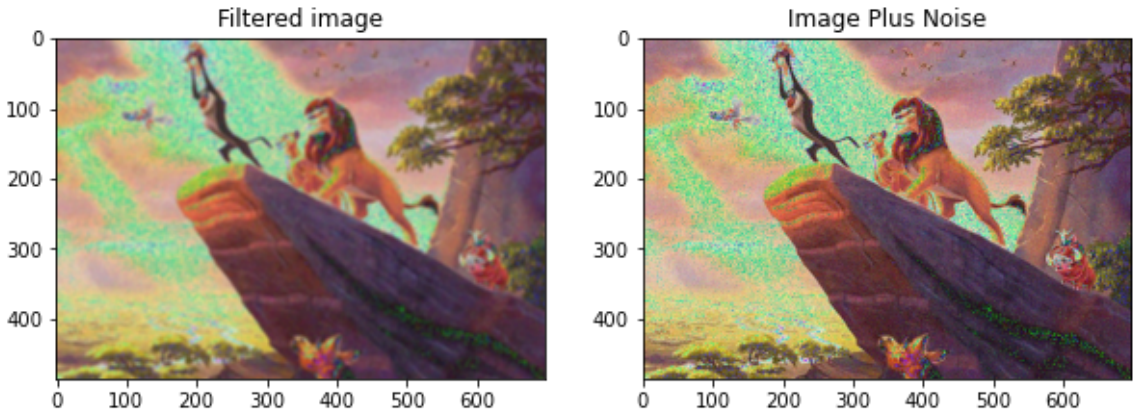
Ввод [49]:

```
# Фильтруем изображения с помощью ядра
image_filtered = cv2.filter2D(src=noisy_image, ddepth=-1, kernel=kernel)
```

Мы можем построить изображение до и после фильтрации; мы видим, что шум уменьшился, но изображение размыто:

Ввод [51]:

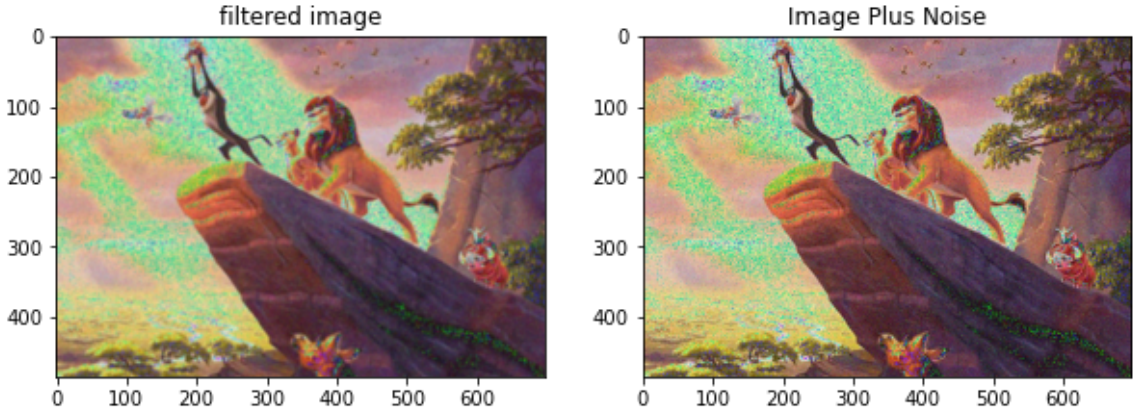
```
# Построим отфильтрованное изображение и изображение с шумом, используя функцию, определе
plot_image(image_filtered, noisy_image,title_1="Filtered image",title_2="Image Plus Noise
```



Ядро меньшего размера сохраняет резкость изображения, но фильтрует меньше шума, здесь мы пробуем ядро 4x4.

Ввод [52]:

```
# Creates a kernel which is a 4 by 4 array where each value is 1/16
kernel = np.ones((4,4))/16
# Filters the images using the kernel
image_filtered=cv2.filter2D(src=noisy_image,ddepth=-1,kernel=kernel)
# Plots the Filtered and Image with Noise using the function defined at the top
plot_image(image_filtered , noisy_image,title_1="filtered image",title_2="Image Plus Nois
```



Размытие по Гауссу

Функция `GaussianBlur` сворачивает исходное изображение с указанным ядром Гаусса. Она фильтрует шум, но лучше сохраняет края. Он имеет следующие параметры:

Parameters

`src` входное изображение; изображение может иметь любое количество каналов, которые обрабатываются независимо

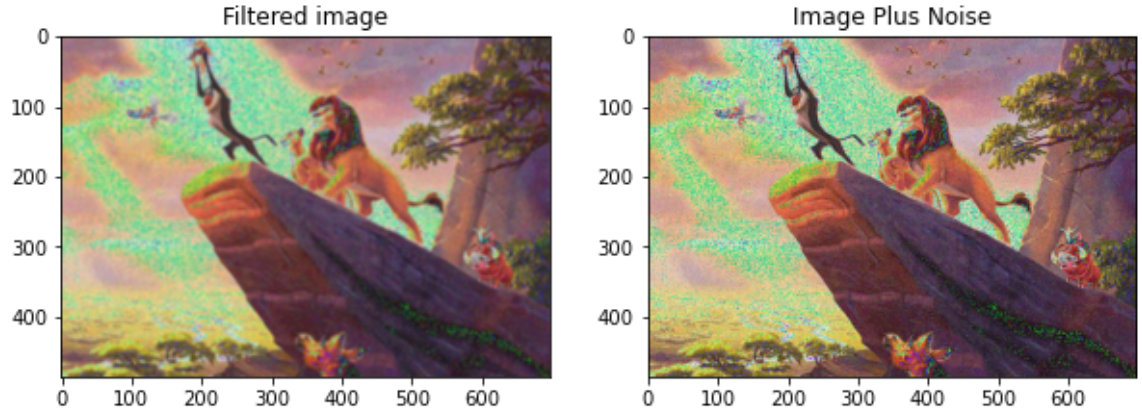
`ksize:` Гауссовский размер ядра

`sigmaX` Стандартное отклонение ядра Гаусса в направлении X

`sigmaY` Стандартное отклонение ядра Гаусса в направлении Y; если sigmaY равно нулю, оно устанавливается равным sigmaX

Ввод [53]:

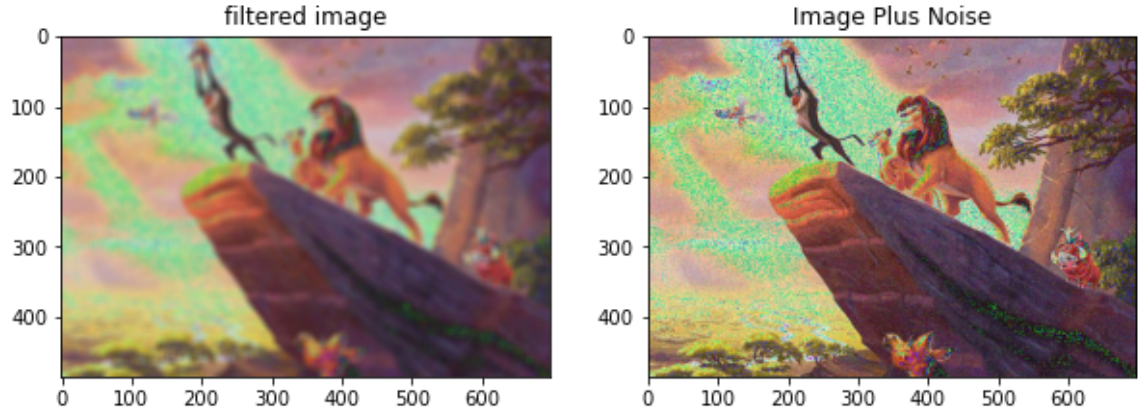
```
# Filters the images using GaussianBlur on the image with noise using a 4 by 4 kernel
image_filtered = cv2.GaussianBlur(noisy_image,(5,5),sigmaX=4,sigmaY=4)
# Plots the Filtered Image then the Unfiltered Image with Noise
plot_image(image_filtered , noisy_image,title_1="Filtered image",title_2="Image Plus Nois
```



Сигма ведет себя как размер среднего фильтра, большее значение сигмы сделает изображение размытым, но вы все еще ограничены размером фильтра, здесь мы устанавливаем сигму на 10

Ввод [54]:

```
# Filters the images using GaussianBlur on the image with noise using a 11 by 11 kernel
image_filtered = cv2.GaussianBlur(noisy_image,(11,11),sigmaX=10,sigmaY=10)
# Plots the Filtered Image then the Unfiltered Image with Noise
plot_image(image_filtered , noisy_image,title_1="filtered image",title_2="Image Plus Nois
```



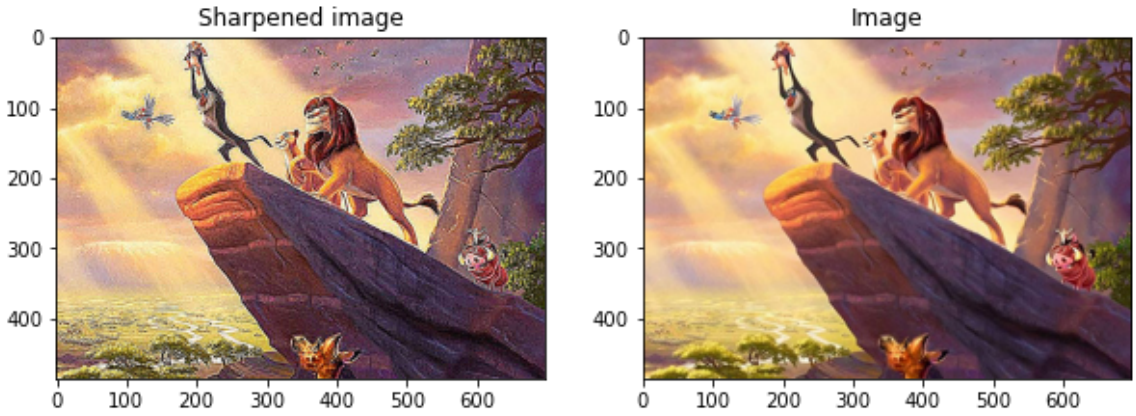
Посмотрите, что происходит, когда вы устанавливаете разные значения sigmaX, sigmaY и/или используете неквадратные ядра.

Повышение резкости изображения

Повышение резкости изображения включает в себя сглаживание изображения и вычисление производных. Мы можем повысить резкость изображения, применив следующее ядро.

Ввод [55]:

```
# Common Kernel for image sharpening
kernel = np.array([[-1,-1,-1],
                   [-1, 9,-1],
                   [-1,-1,-1]])
# Applys the sharpening filter using kernel on the original image without noise
sharpened = cv2.filter2D(image, -1, kernel)
# Plots the sharpened image and the original image without noise
plot_image(sharpened , image, title_1="Sharpened image",title_2="Image")
```



Края (Edges)

Края — это места, где меняется интенсивность пикселей. Градиент функции выводит скорость изменения. Можно аппроксимировать градиент изображения в градациях серого с помощью свертки. Существует несколько методов аппроксимации градиента, давайте воспользуемся детектором границ Собеля. Это объединяет несколько сверток и нахождение величины результата. Рассмотрим следующее изображение:

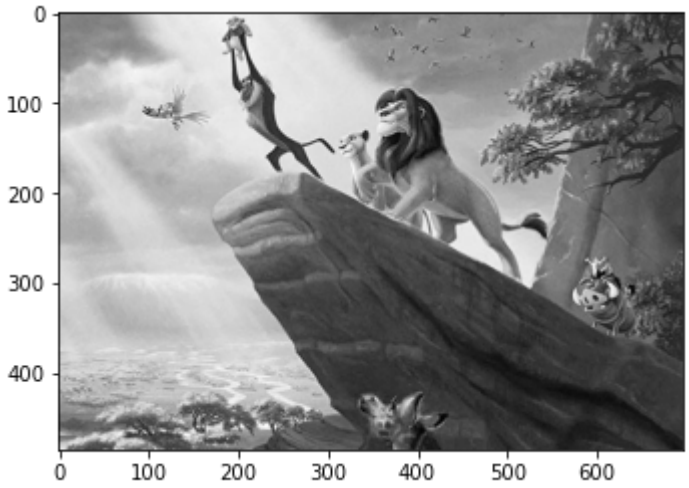
Ввод [56]:

```
# Loads the image from the specified file
img_gray = cv2.imread('2.png', cv2.IMREAD_GRAYSCALE)
print(img_gray)
# Renders the image from the array of data, notice how it is 2 diemensional instead of 3
plt.imshow(img_gray ,cmap='gray')
```

```
[[141 141 141 ... 105  94  80]
 [141 142 141 ...  81  75  69]
 [141 141 140 ...  57  59  55]
 ...
 [219 217 222 ...  27  29  31]
 [229 226 229 ...  24  26  29]
 [231 229 230 ...  25  26  31]]
```

Out[56]:

```
<matplotlib.image.AxesImage at 0x1c336b12b50>
```



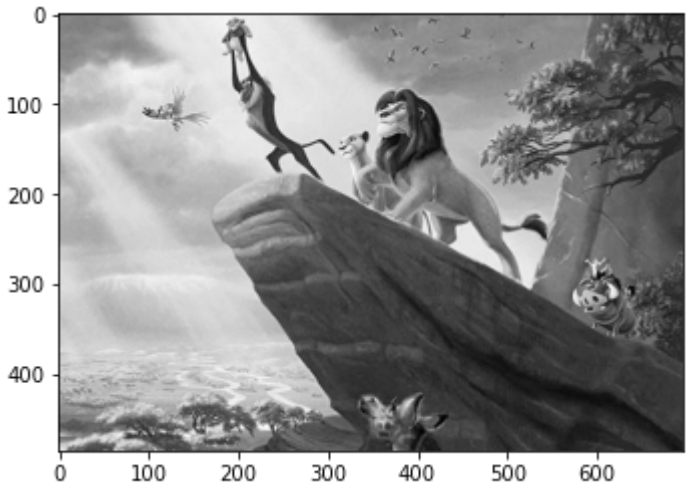
Мы сглаживаем изображение, это уменьшает изменения, которые могут быть вызваны шумом, влияющим на градиент.

Ввод [57]:

```
# Filters the images using GaussianBlur on the image with noise using a 3 by 3 kernel
img_gray = cv2.GaussianBlur(img_gray,(3,3),sigmaX=0.1,sigmaY=0.1)
# Renders the filtered image
plt.imshow(img_gray ,cmap='gray')
```

Out[57]:

```
<matplotlib.image.AxesImage at 0x1c336c5dbe0>
```



Мы можем аппроксимировать производную в направлении X или Y, используя функцию `Sobel` , вот параметры:

`src` : входное изображение

`ddepth` : глубина вывода изображения; в случае 8-битных входных изображений это приведет к усеченным производным

`dx` : порядок производной x

`dx` : порядок производной y

`ksize` размер расширенного ядра Sobel; должно быть 1, 3, 5 или 7

dx = 1 представляет собой производную по оси x. Функция аппроксимирует производную путем свертки изображения со следующим ядром

\begin{bmatrix} 1 & 0 & -1 \\\\ 2 & 0 & -2 \\\\ 1 & 0 & -1 \end{bmatrix}

Мы можем применить функцию:

Ввод [59]:

```
ddepth = cv2.CV_16S
# Applys the filter on the image in the X direction
grad_x = cv2.Sobel(src=img_gray, ddepth=ddepth, dx=1, dy=0, ksize=3)
```

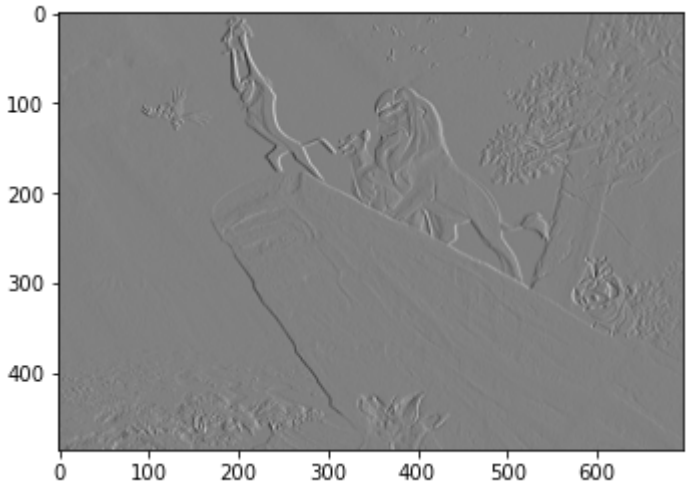
Мы можем построить результат

Ввод [60]:

```
plt.imshow(grad_x,cmap='gray')
```

Out[60]:

```
<matplotlib.image.AxesImage at 0x1c33699a9a0>
```



dy=1 представляет собой производную в направлении y. Функция аппроксимирует производную путем свертки изображения со следующим ядром

\begin{bmatrix} \ \ 1 & \ \ 2 & \ \ 1 \\\\ \ \ 0 & \ \ 0 & \ \ 0 \\\\ -1 & -2 & -1 \end{bmatrix}

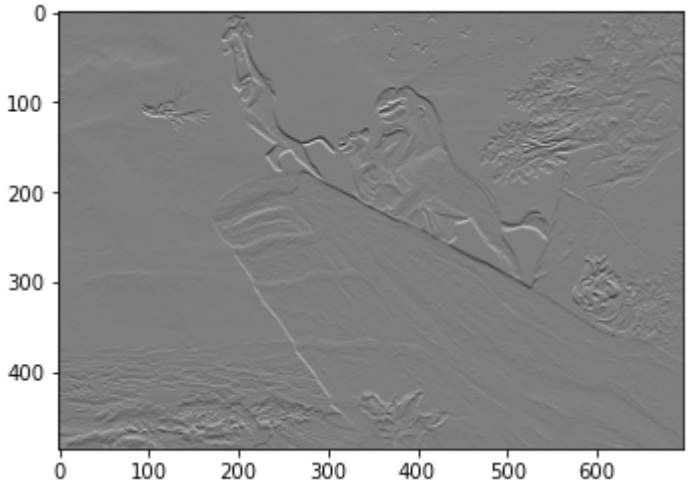
Мы можем применить функцию и построить график результата

Ввод [61]:

```
# Applys the filter on the image in the X direction
grad_y = cv2.Sobel(src=img_gray, ddepth=ddepth, dx=0, dy=1, ksize=3)
plt.imshow(grad_y,cmap='gray')
```

Out[61]:

```
<matplotlib.image.AxesImage at 0x1c33696b220>
```



Мы можем аппроксимировать градиент, вычислив абсолютные значения и преобразовав результат в 8-битный:

Ввод [62]:

```
# Converts the values back to a number between 0 and 255
abs_grad_x = cv2.convertScaleAbs(grad_x)
abs_grad_y = cv2.convertScaleAbs(grad_y)
```

Затем примените функцию `addWeighted` для вычисления суммы двух массивов следующим образом:

Ввод [63]:

```
# Adds the derivative in the X and Y direction
grad = cv2.addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0)
```

Затем мы наносим результаты на график, мы видим, что изображение с линиями имеет значения высокой интенсивности, представляющие большой градиент.

Ввод [64]:

```
# Make the figure bigger and renders the image
plt.figure(figsize=(10,10))
plt.imshow(grad,cmap='gray')
```

Out[64]:

```
<matplotlib.image.AxesImage at 0x1c3369b25b0>
```

Медиана

Медианные фильтры находят медиану всех пикселей под областью ядра, и центральный элемент заменяется этим медианным значением.

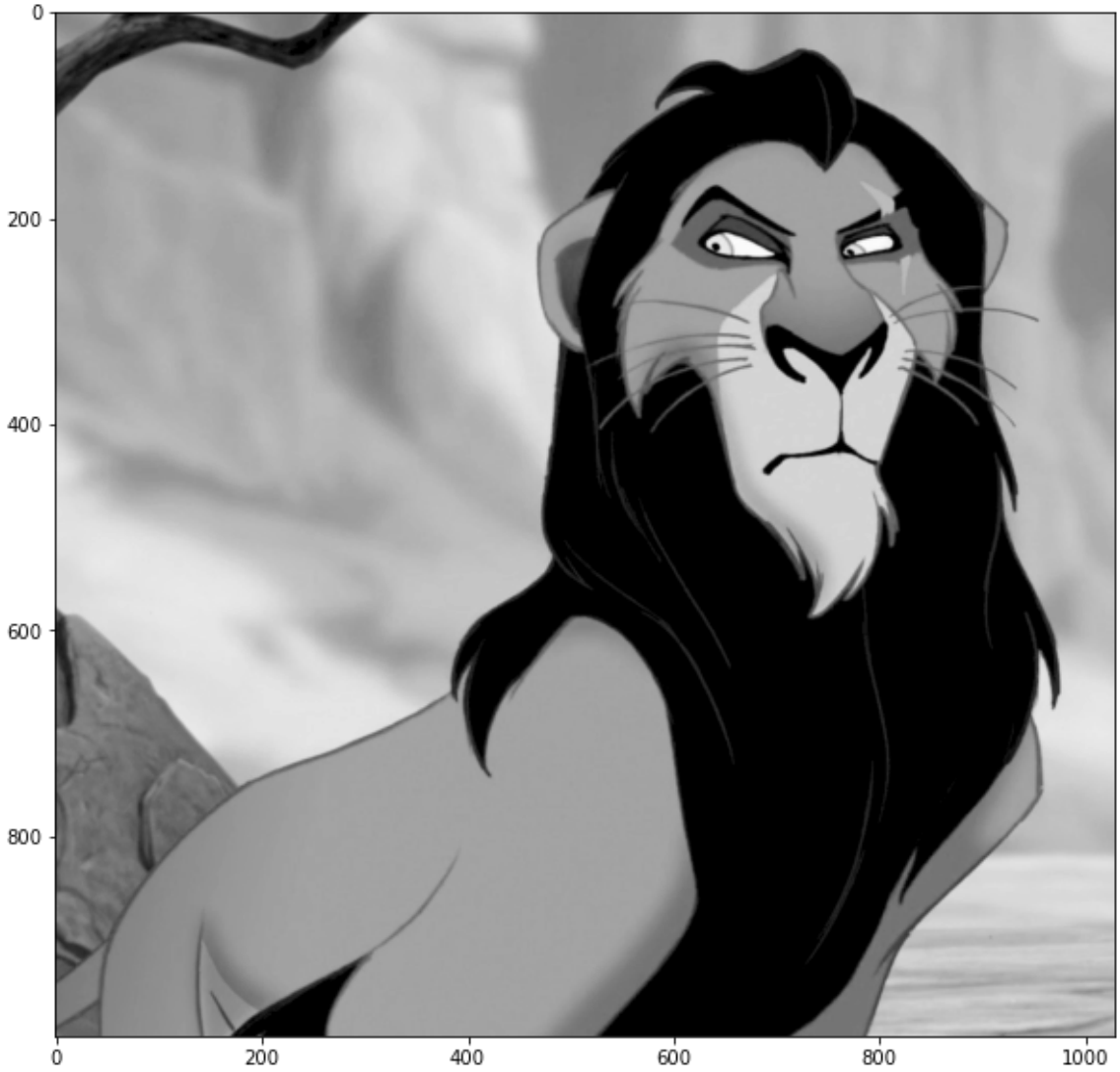
Мы можем применять медианные фильтры к обычным изображениям, но давайте посмотрим, как мы можем использовать медианные фильтры для улучшения сегментации.

Ввод [65]:

```
# Load the camera man image
image = cv2.imread("3.jpeg",cv2.IMREAD_GRAYSCALE)
# Make the image larger when it renders
plt.figure(figsize=(10,10))
# Renders the image
plt.imshow(image,cmap="gray")
```

Out[65]:

```
<matplotlib.image.AxesImage at 0x1c3372f8160>
```



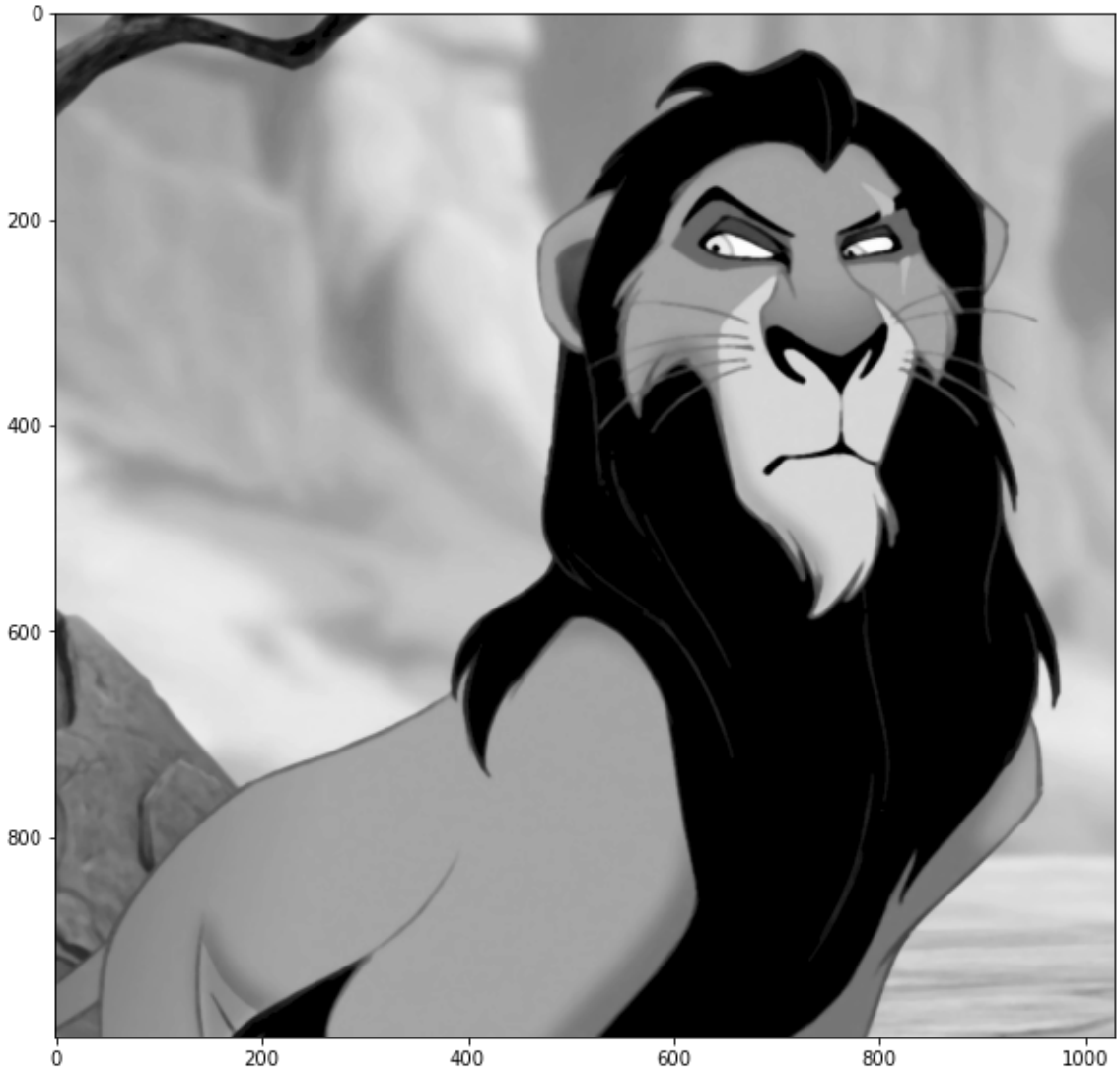
Теперь давайте применим медианный фильтр с помощью функции `medianBlur` . Параметры этой функции: `src` : изображение и `ksize` : размер ядра.

Ввод [66]:

```
# Filter the image using Median Blur with a kernel of size 5
filtered_image = cv2.medianBlur(image, 5)
# Make the image larger when it renders
plt.figure(figsize=(10,10))
# Renders the image
plt.imshow(filtered_image,cmap="gray")
```

Out[66]:

```
<matplotlib.image.AxesImage at 0x1c336bd0580>
```



Параметры пороговой функции

`src` : изображение для использования

`thresh` : порог

`maxval` : максимальное значение для использования

`type` : Тип фильтрации

Пороговая функция работает, просматривая значение оттенков серого каждого пикселя и присваивая значение, если оно ниже порога, и другое значение, если оно выше порога. В нашем примере порог равен 0 (черный), а тип — двоичный инверсный, поэтому, если значение выше порога, присвоенное значение равно 0 (черный), а если оно ниже или равно порогу, используется maxval 255 (белый). Таким образом, если пиксель 0 черный, ему назначается 255 (белый), а если пиксель не черный, то он назначается черным, что `THRESH_BINARY_INV` сообщает `OpenCV` . Вот как это будет работать без `THRESH_OTSU` .

Поскольку мы используем `THRESH_OTSU` , это означает, что `OpenCV` определит оптимальный порог.

Ввод [67]:

```
# Returns ret which is the threshold used and outs which is the image
ret, outs = cv2.threshold(src = image, thresh = 0, maxval = 255, type = cv2.THRESH_OTSU+c

# Make the image larger when it renders
plt.figure(figsize=(10,10))

# Render the image
plt.imshow(outs, cmap='gray')
```

Out[67]:

```
<matplotlib.image.AxesImage at 0x1c33699f3d0>
```

Практическая работа 3 "Pillow Library (PIL)"

1. Задание: Необходимо повторить весь код из примера с одним своим изображением, что означает, что все манипуляции должны быть произведены с одним изображением.
2. Прикрепить файл в формате .ipynb.
3. Балл за задание: 3.

Немного вводной части

Задачи обработки изображений и компьютерного зрения включают отображение, обрезку, переворачивание, поворот, сегментацию изображения, классификацию, восстановление изображения, распознавание изображений, создание изображений. Кроме того, иногда работа с изображениями требует хранения, передачи и сбора изображений через Интернет.

Python - отличный выбор, поскольку в нем есть множество инструментов для обработки изображений, компьютерного зрения и библиотек искусственного интеллекта. Также, в нем есть множество библиотек для работы с файлами в облачных сервисах и в сети Интернете. Цифровое изображение - это просто файл на вашем компьютере. В данной практической работе вы получите представление об этих файлах и научитесь работать с ними с помощью библиотеки Pillow (PIL).

Загрузка изображений для практической работы:

Ввод [1]:

```
from IPython.display import Image
Image("pika.jpg")
```

Out[1]:

Для начала, необходимо определить вспомогательную функцию для объединения двух изображений:

Ввод [2]:

```
def get_concat_h(im1, im2):
    #https://note.nkmk.me/en/python-pillow-concat-images/
    dst = Image.new('RGB', (im1.width + im2.width, im1.height))
    dst.paste(im1, (0, 0))
    dst.paste(im2, (im1.width, 0))
    return dst
```

Изображения и пути хранения

Изображение хранится в виде файла на вашем компьютере. Ниже мы определяем my_image как имя файла в этом каталоге.

Ввод [3]:

```
my_image = "pika.jpg"
```

Имя файла состоит из двух частей: имени файла и расширения, разделенных точкой (`.`). Расширение определяет формат изображения. Существует два популярных формата изображений: изображение Joint Photographic Expert Group (или .jpg, .jpeg) и Portable Network Graphics (или .png). Эти типы файлов упрощают работу с изображениями. Например, они могут сжимать изображение, что позволяет занимать меньше места для хранения изображения.

Файлы изображений хранятся в файловой системе вашего компьютера. Его местоположение указывается с помощью уникального «пути». Вы можете найти путь к вашему текущему рабочему каталогу с помощью модуля Python `os`. Модуль `os` предоставляет функции для взаимодействия с файловой системой, например, создание или удаление каталога (папки), перечисление ее содержимого, изменение и идентификация текущего рабочего каталога.

Ввод [4]:

```
import os
cwd = os.getcwd()
cwd
```

Out[4]:

```
'C:\\Users\\ppblondy\\Dropbox\\Универ лекции\\2022-2023\\1 семестр\\Поиск
и обработка\\Практическая_работа_№_1_Базовые манипуляции в библиотеке Pill
ow'
```

Путь к изображению можно найти с помощью следующей строки кода

Ввод [5]:

```
image_path = os.path.join(cwd, my_image)
image_path
```

Out[5]:

```
'C:\\Users\\ppblondy\\Dropbox\\Универ лекции\\2022-2023\\1 семестр\\Поиск
и обработка\\Практическая_работа_№_1_Базовые манипуляции в библиотеке Pill
ow\\pika.jpg'
```

Загрузка изображений

Библиотека Pillow (PIL) - популярная библиотека для загрузки изображений в Python. Кроме того, многие другие библиотеки, такие как «Keras» и «PyTorch», используют эту библиотеку для работы с изображениями. Модуль `Image` предоставляет функции для загрузки изображений и сохранения изображений в файловой системе. Импортируем его из `PIL` .

Ввод [7]:

```
from PIL import Image
```

Если изображение находится в текущем рабочем каталоге, вы можете загрузить изображение следующим образом, используя имя файла изображения и создать объект PIL Image:

Ввод [10]:

```
image = Image.open(my_image)
type(image)
```

Out[10]:

```
PIL.JpegImagePlugin.JpegImageFile
```

Если вы работаете в среде Jupyter, вы можете просмотреть изображение, вызвав саму переменную:

Ввод [9]:

```
image
```

Out[9]:

Отрисовка изображени

Мы также можем использовать метод `show` объектов PIL для отображения изображения. Обратите внимание, что этот метод может работать или не работать в зависимости от ваших настроек.

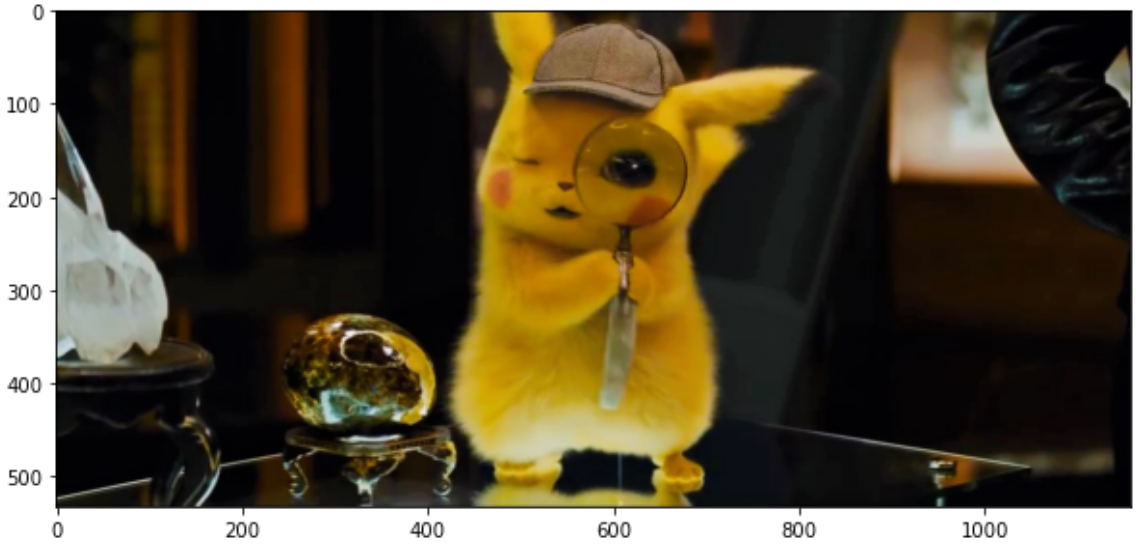
Ввод [11]:

```
image.show()
```

Вы также можете использовать метод imshow из библиотеки matplotlib для отображения изображения.

Ввод [13]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,10))
plt.imshow(image)
plt.show()
```



Вы также можете загрузить изображение, используя его полный путь. Это пригодится, если изображение отсутствует в вашем рабочем каталоге.

Ввод [14]:

```
image = Image.open(image_path)
```

Вы можете использовать атрибуты объекта изображения для получения информации. Формат атрибута - это расширение или формат изображения.

Атрибут size возвращает кортеж, первый элемент - это количество пикселей, составляющих ширину, а второй элемент - количество пикселей, составляющих высоту изображения.

Ввод [15]:

```
print(image.size)
```

```
(1158, 533)
```

Ниже приведена строка, определяющая используемый формат пикселей. В данном случае это «RGB». RGB - это цветовое пространство, в котором красный, зеленый и синий складываются вместе для получения других цветов.

Ввод [16]:

```
print(image.mode)
```

```
RGB
```

Метод Image.open не загружает данные изображения в память компьютера. Метод load объекта PIL считывает содержимое файла, декодирует его и расширяет изображение в память.

Ввод [17]:

```
im = image.load()
```

Затем мы можем проверить интенсивность изображения в x -м столбце и y -й строке:

Ввод [18]:

```
x = 0
y = 1
im[y,x]
```

Out[18]:

```
(2, 2, 4)
```

Мы будем использовать форму массива для доступа к элементам.

Вы можете сохранить изображение в формате `jpg` , используя следующую команду:

Ввод [20]:

```
image.save("Pika.jpg")
```

Изображения в оттенках серого, квантование и цветовые каналы

Изображения в оттенках серого

Модуль ImageOps содержит несколько готовых операций обработки изображений. Большинство операторов данного модуля работают только с изображениями в оттенках серого и/или RGB.

Ввод [21]:

```
from PIL import ImageOps
```

Изображения в градациях серого имеют значения пикселей, представляющие количество света или интенсивность этого пикселя. Светлые оттенки серого имеют высокую интенсивность, тогда как более темные оттенки имеют более низкую интенсивность, то есть белый имеет самую высокую интенсивность, а черный - самую низкую.

Ввод [22]:

```
image_gray = ImageOps.grayscale(image)
image_gray
```

Out[22]:

Режим `L` для оттенков серого.

Ввод [23]:

```
image_gray.mode
```

Out[23]:

```
'L'
```

Квантование

Квантование изображения - это количество уникальных значений интенсивности, которые может принимать любой заданный пиксель изображения. Для изображения в градациях серого это означает количество различных оттенков серого. Большинство изображений имеют 256 различных уровней. Можно уменьшить уровни, используя метод «quantize». Рассмотрим пример с сокращением количества уровней вдвое.

Половина уровней не дает заметной разницы.

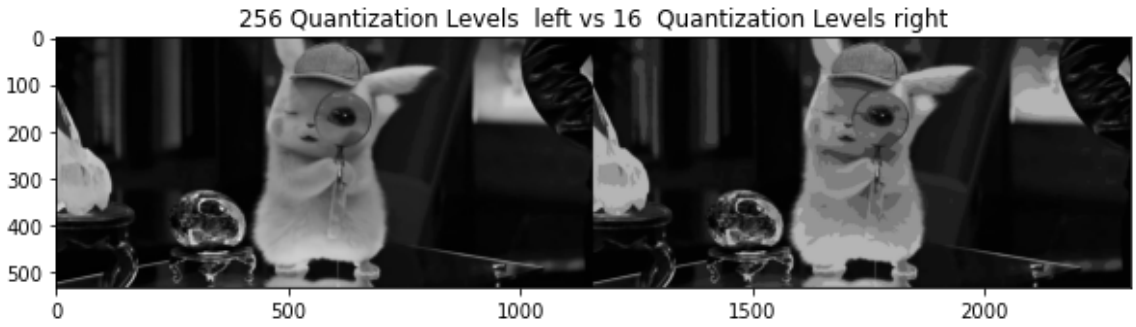
Ввод [24]:

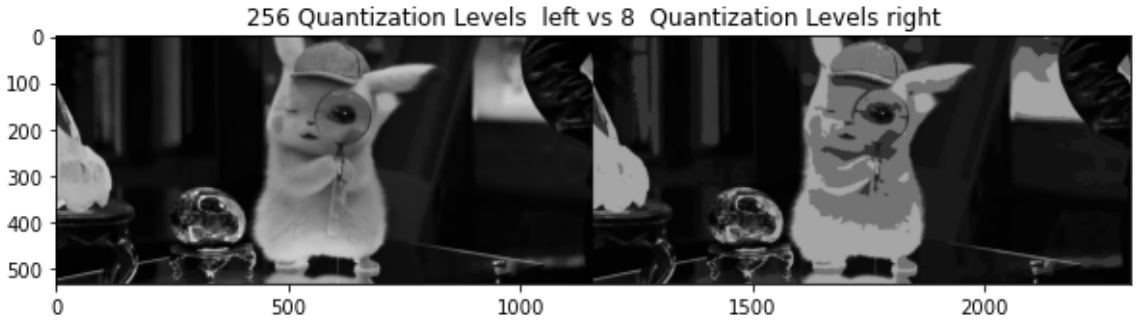
```
image_gray.quantize(256 // 2)
image_gray.show()
```

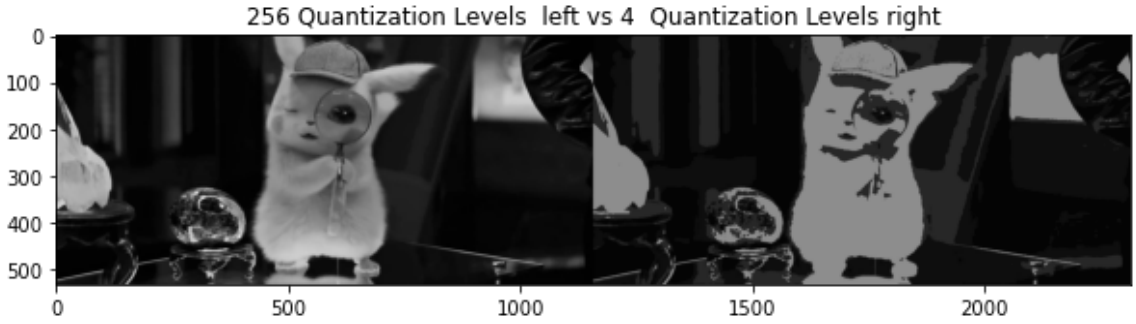
Рассмотрим деление количества значений на два и сравним его с исходным изображением.

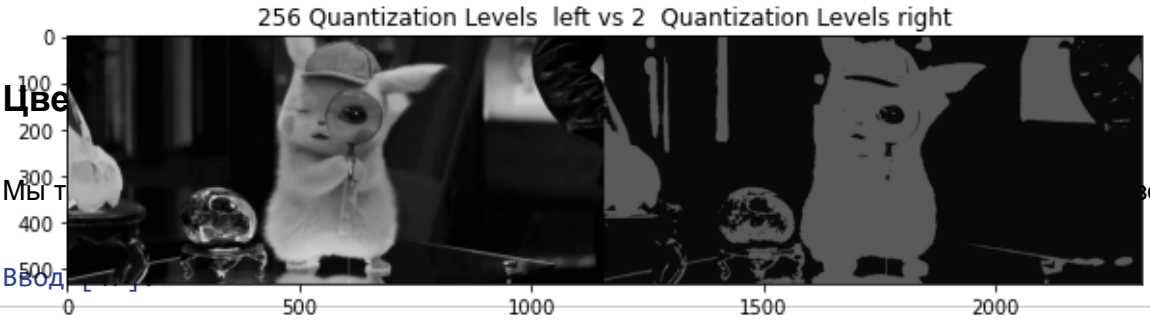
Ввод [25]:

```
#get_concat_h(image_gray,  image_gray.quantize(256//2)).show(title="Pika")
for n in range(3,8):
    plt.figure(figsize=(10,10))

    plt.imshow(get_concat_h(image_gray,  image_gray.quantize(256//2**n)))
    plt.title("256 Quantization Levels  left vs {}  Quantization Levels right".format(256
    plt.show()
```









Цве

Мы т зображение:

Ввод [47]:

```
from IPython.display import Image
Image("Mushu.jpg")
```

Out[47]:

Ввод [48]:

```
mushu = "Mushu.jpg"
```

Ввод [49]:

```
from PIL import Image
```

Ввод [50]:

```
Mushu = Image.open(mushu)
type(Mushu)
```

Out[50]:

```
PIL.JpegImagePlugin.JpegImageFile
```

Ввод [51]:

```
Mushu
```

Out[51]:

Можно получить различные цветовые каналы RGB и назначить их переменным red , green и blue :

Ввод [52]:

```
red, green, blue = Mushu.split()
```

Помещая цветное изображение рядом с красным каналом в градациях серого, можно увидеть, что области с красным цветом имеют более высокие значения интенсивности.

Ввод [33]:

```
get_concat_h(Mushu, red)
```

Out[33]:

Ввод [34]:

```
get_concat_h(Mushu, blue)
```

Out[34]:

Ввод [35]:

```
get_concat_h(Mushu, green)
```

Out[35]:

Изображения PIL в массивах NumPy

NumPy - это библиотека Python, позволяющая работать с многомерными массивами и матрицами. Можно преобразовать изображение PIL в массив NumPy. Используем asarray() или функцию массива из NumPy для преобразования изображений PIL в массивы NumPy. Для начала, импортируем модуль numpy:

Ввод [36]:

```
import numpy as np
```

Применяем его к изображению PIL, получаем массив numpy:

Ввод [53]:

```
array= np.asarray(Mushu)
print(type(array))
```

```
<class 'numpy.ndarray'>
```

np.asarray превращает исходное изображение в массив numpy. Очень часто необходимо не напрямую манипулировать изображением. В этом случае необходимо создать копию изображения для дальнейших манипуляций. Метод `np.array` создает новую копию изображения, так что исходная остается неизменной.

Ввод [54]:

```
array = np.array(Mushu)
```

Атрибут `shape` объекта `numpy.array` возвращает кортеж, соответствующий его размерам, первый элемент дает количество строк или высоту изображения, второй элемент - это количество столбцы или ширина изображения. Последний элемент - количество цветовых каналов.

Ввод [55]:

```
# summarize shape
print(array.shape)
```

```
(546, 728, 3)
```

или (строки, столбцы, цвета). Каждый элемент на оси цвета соответствует следующему формату значений (R, G, B).

Мы можем просмотреть значения интенсивности, распечатав массив, они варьируются от 0 до 255 или 28 (8-битные).

Ввод [56]:

```
print(array)
```

```
[[[ 88 111 181]
  [ 88 111 181]
  [ 88 111 181]
  ...
  [  4   6   5]
  [  4   6   5]
  [  4   6   5]]

 [[ 88 111 181]
  [ 88 111 181]
  [ 88 111 181]
  ...
  [  4   6   5]
  [  4   6   5]
  [  4   6   5]]

 [[ 88 111 181]
  [ 88 111 181]
  [ 88 111 181]
  ...
  [  4   6   5]
  [  4   6   5]
  [  4   6   5]]

 ...

 [[124 175 222]
  [125 176 223]
  [126 177 224]
  ...
  [ 56  67  51]
  [ 56  67  51]
  [ 56  67  51]]

 [[125 176 221]
  [125 176 221]
  [126 177 222]
  ...
  [ 57  68  52]
  [ 57  68  52]
  [ 57  68  52]]

 [[125 176 221]
  [125 176 221]
  [126 177 222]
  ...
  [ 57  68  52]
  [ 57  68  52]
  [ 57  68  52]]]
```

Значения интенсивности - это 8-битные беззнаковые данные.

Ввод [57]:

```
array[0, 0]
```

Out[57]:

```
array([ 88, 111, 181], dtype=uint8)
```

Мы можем найти максимальное и минимальное значение интенсивности массива:

Ввод [58]:

```
array.min()
```

Out[58]:

```
0
```

Ввод [59]:

```
array.max()
```

Out[59]:

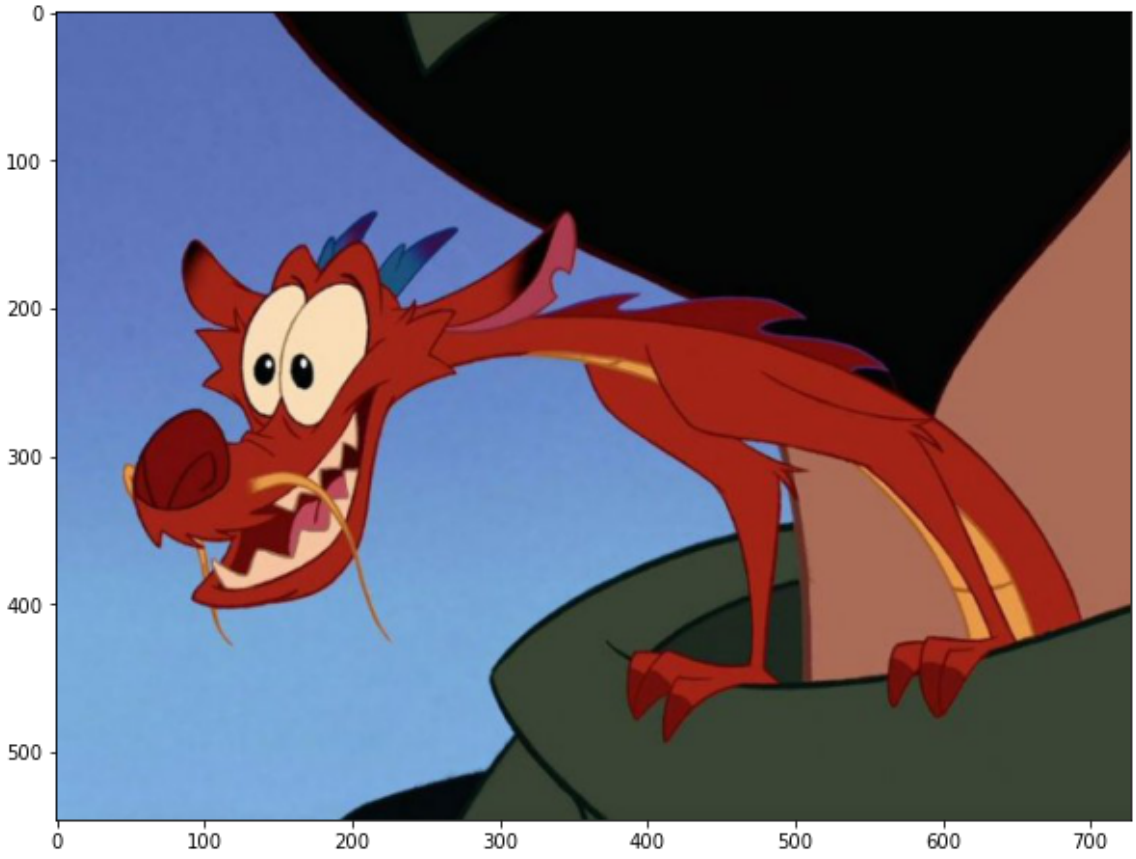
```
255
```

Индексирование

Вы можете построить массив как изображение:

Ввод [60]:

```
plt.figure(figsize=(10,10))
plt.imshow(array)
plt.show()
```



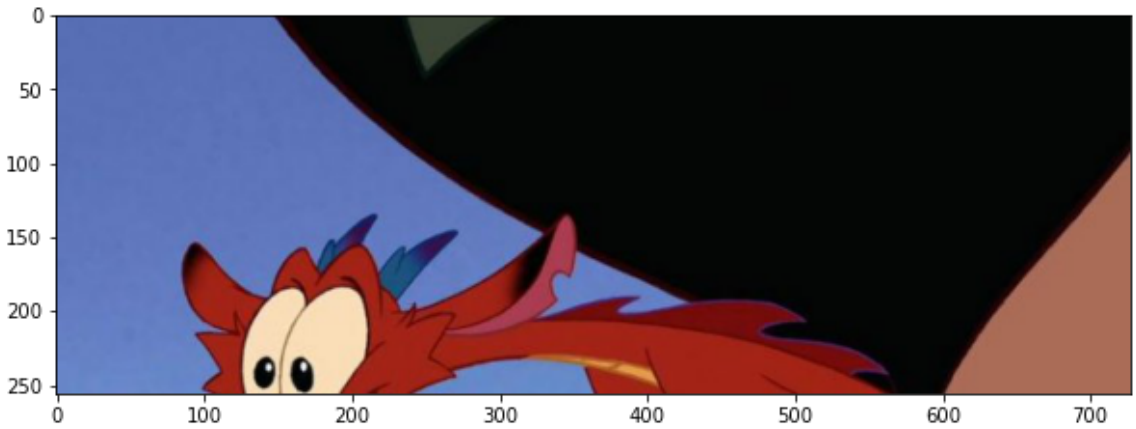
Вы можете использовать нарезку numpy, например, можно вернуть первые 256 строк, соответствующих верхней половине изображения:

Ввод [61]:

```
rows = 256
```

Ввод [62]:

```
plt.figure(figsize=(10,10))
plt.imshow(array[0:rows,:,:])
plt.show()
```



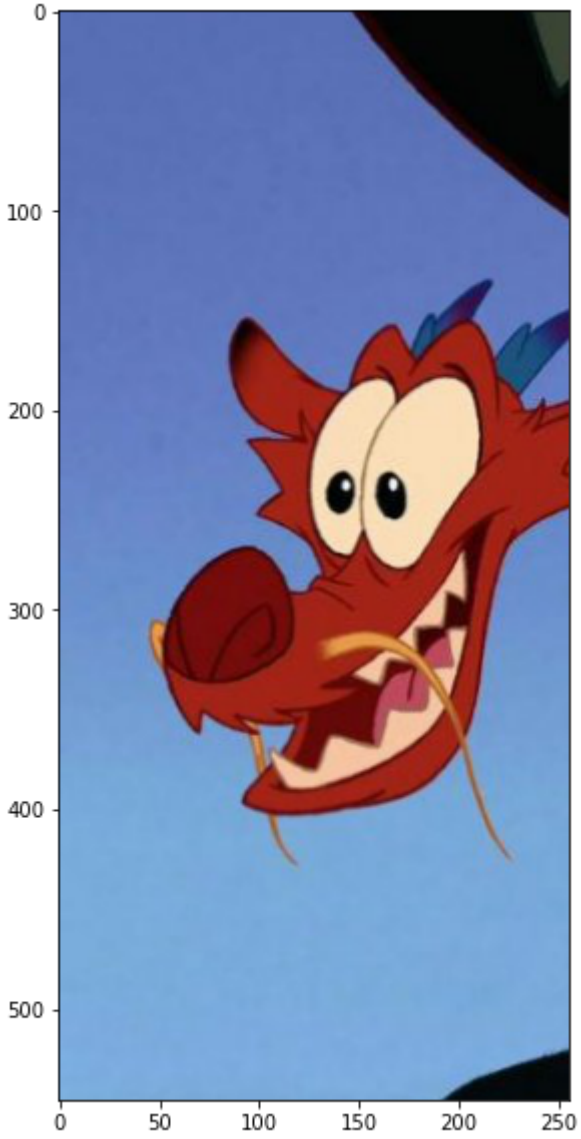
Можно вернуть первые 256 столбцов, соответствующих первой половине изображения.

Ввод [63]:

```
columns = 256
```

Ввод [64]:

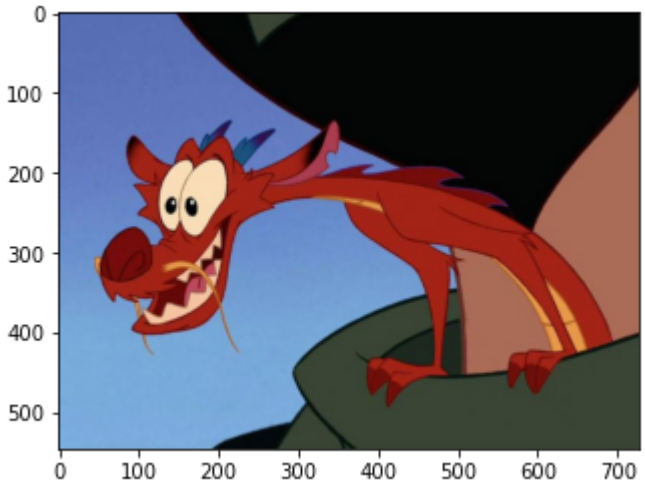
```
plt.figure(figsize=(10,10))
plt.imshow(array[:,0:columns,:])
plt.show()
```



Если вы хотите переназначить массив другой переменной, вы должны использовать метод `copy` .

Ввод [65]:

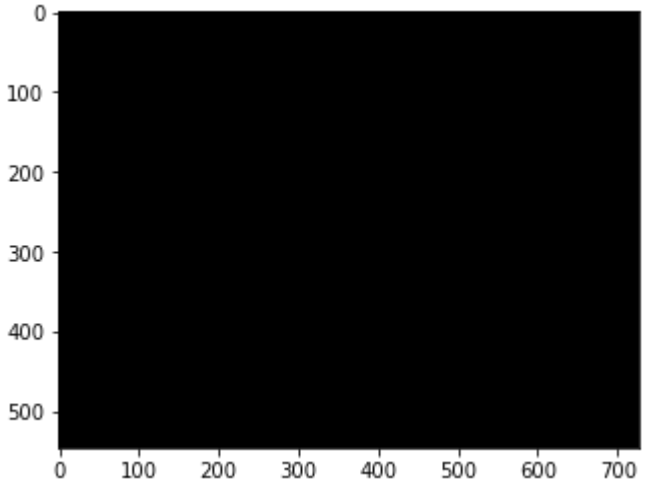
```
A = array.copy()
plt.imshow(A)
plt.show()
```



Если мы не применим метод copy (), переменная будет указывать на то же место в памяти. Рассмотрим массив B. Если мы установим все значения массива A равными нулю, поскольку B указывает на A, значения B тоже будут равны нулю:

Ввод [67]:

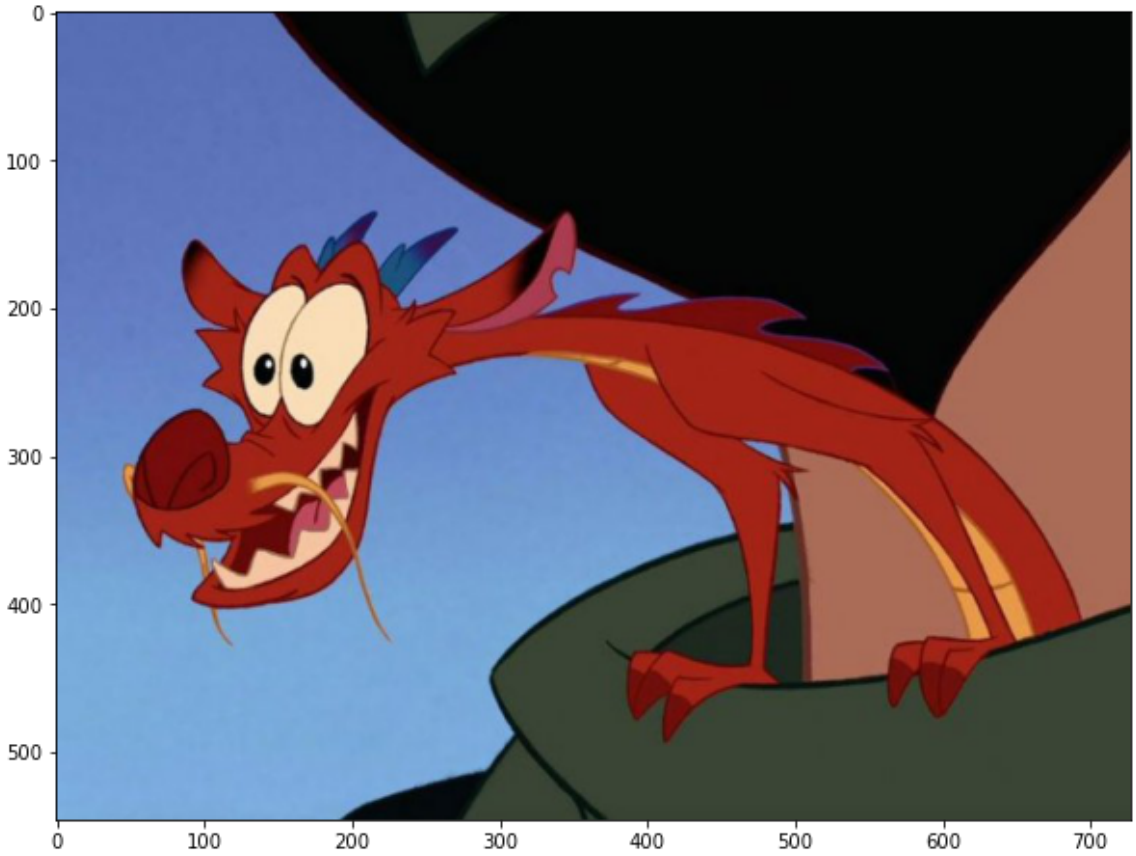
```
B = A
A[:,:,:] = 0
plt.imshow(B)
plt.show()
```



Также можно работать с разными цветовыми каналами. Рассмотрим изображение:

Ввод [68]:

```
Mushu_array = np.array(Mushu)
plt.figure(figsize=(10,10))
plt.imshow(Mushu_array)
plt.show()
```



Можно построить красный канал как значения интенсивности красного канала.

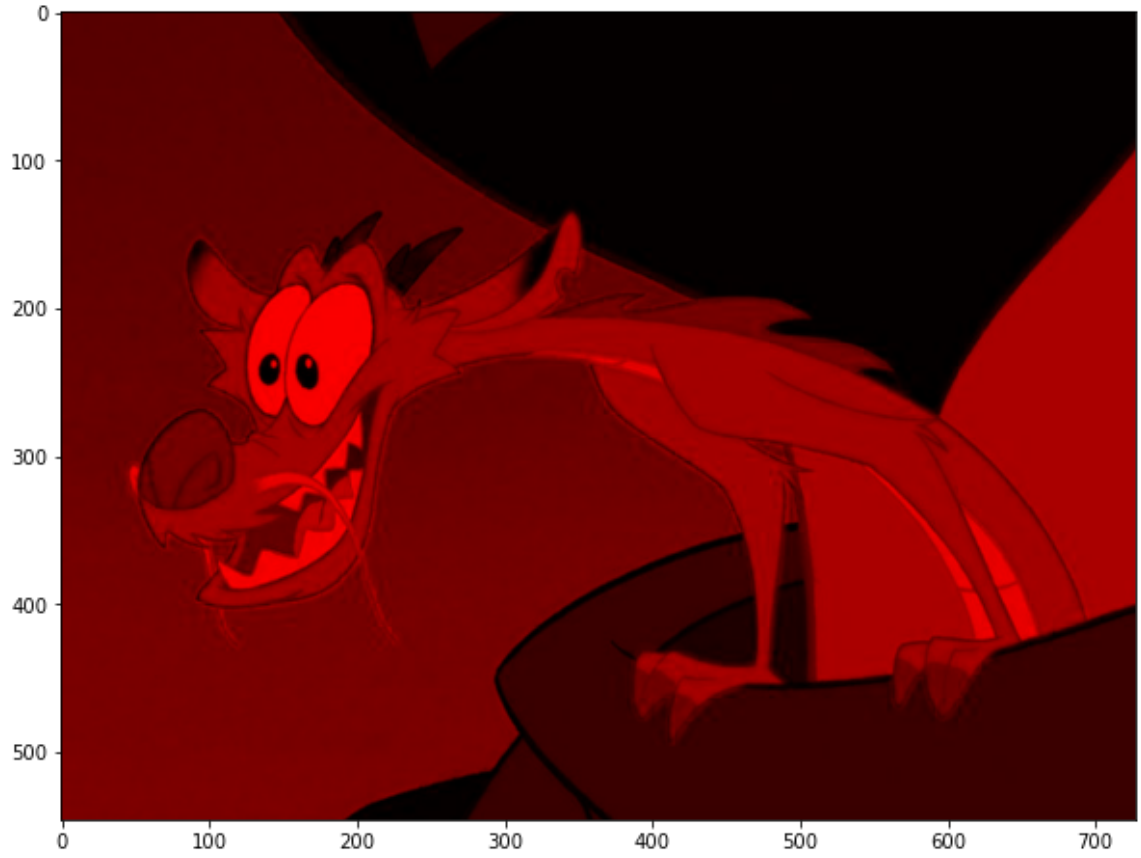
Ввод [69]:

```
Mushu_array = np.array(Mushu)
plt.figure(figsize=(10,10))
plt.imshow(Mushu_array[:,:,0], cmap='gray')
plt.show()
```

Можно создать новый массив и обнулить все каналы, кроме красного.

Ввод [70]:

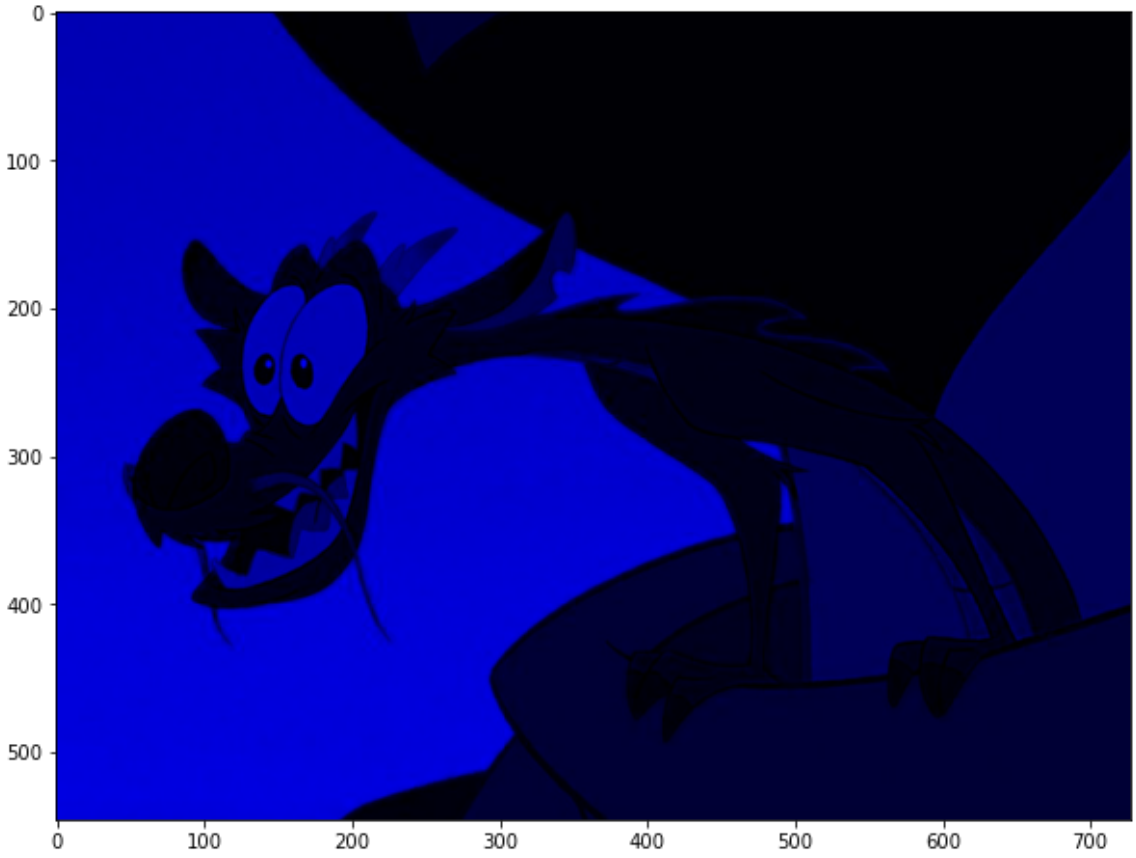
```
Mushu_red = Mushu_array.copy()
Mushu_red[:,:,1] = 0
Mushu_red[:,:,2] = 0
plt.figure(figsize=(10,10))
plt.imshow(Mushu_red)
plt.show()
```



То же самое можно сделать и с синим:

Ввод [71]:

```
Mushu_blue = Mushu_array.copy()
Mushu_blue[:,:,0] = 0
Mushu_blue[:,:,1] = 0
plt.figure(figsize=(10,10))
plt.imshow(Mushu_blue)
plt.show()
```



Рассмотрим на примере преобразование изображения в numpy-массив, извлечем из него синий канал и построим изображение.

Ввод [74]:

```
Mushu_blue[:,:,2] = 0
```

Ввод [75]:

```
blue_Mushu = Image.open('Mushu.jpg')
blue_array = np.array(blue_Mushu)
blue_array[:,:,2] = 0
plt.figure(figsize=(10,10))
plt.imshow(blue_array)
plt.show()
```

Практическая работа 4 "Геометрические трансформации и специальные функции в библиотеке Pillow"

1. Задание: Необходимо повторить весь код из примера с одним своим изображением, что означает, что все манипуляции должны быть произведены с одним изображением.
2. Прикрепить файл в формате .ipynb.
3. Балл за задание: 3.

Введение

Вы научитесь применять к изображению геометрические преобразования. Это позволяет выполнять различные операции, такие как преобразование трансляции, то есть сдвиг, изменение формы и поворот изображения. Также вы узнаете, как применять к изображению некоторые базовые операции с массивами и матрицами.

Загрузка изображений для практической работы:

Ввод [2]:

```
from IPython.display import Image
Image("1.png")
```

Out[2]:

Библиотечки:

Ввод [3]:

```
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
```

Во-первых, необходимо определить вспомогательную функцию для построения двух изображений бок о бок.

Ввод [4]:

```
def plot_image(image_1, image_2,title_1="Orignal",title_2="New Image"):
    plt.figure(figsize=(10,10))
    plt.subplot(1, 2, 1)
    plt.imshow(image_1,cmap="gray")
    plt.title(title_1)
    plt.subplot(1, 2, 2)
    plt.imshow(image_2,cmap="gray")
    plt.title(title_2)
    plt.show()
```

Геометрические преобразования

Геометрические преобразования позволяют выполнять различные операции, такие как сдвиг, изменение формы и вращение изображения.

Вы можете изменить размер изображения с помощью метода `resize()` из `PIL` , который принимает в качестве параметров изображение с измененным размером `width` и `height` .

Рассмотрим следующее изображение:

Ввод [5]:

```
image = Image.open("1.png")
plt.imshow(image)
plt.show()
```

Вы можете масштабировать горизонтальную ось на два и оставить вертикальную ось как есть:

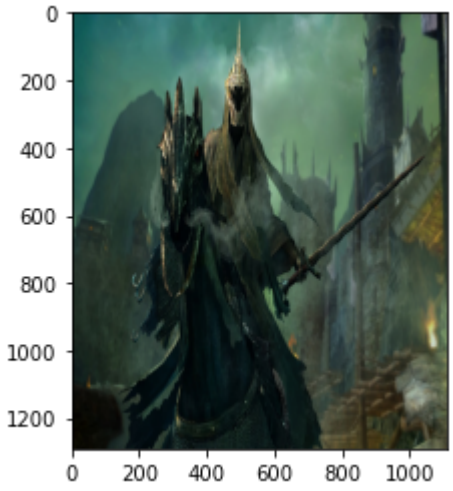
Ввод [6]:

```
width, height = image.size
new_width = 2 * width
new_hight = height
new_image = image.resize((new_width, new_hight))
plt.imshow(new_image)
plt.show()
```

Таким же образом вы можете масштабировать вертикальную ось на два:

Ввод [7]:

```
new_width = width
new_hight = 2 * height
new_image = image.resize((new_width, new_hight))
plt.imshow(new_image)
plt.show()
```



Вы можете удвоить ширину и высоту изображения:

Ввод [8]:

```
new_width = 2 * width
new_hight = 2 * height
new_image = image.resize((new_width, new_hight))
plt.imshow(new_image)
plt.show()
```

Вы также можете уменьшить ширину и высоту изображения на 1/2:

Ввод [9]:

```
new_width = width // 2
new_hight = height // 2

new_image = image.resize((new_width, new_hight))
plt.imshow(new_image)
plt.show()
```

Вращение

Мы можем повернуть изображение на угол θ, используя метод `rotate` .

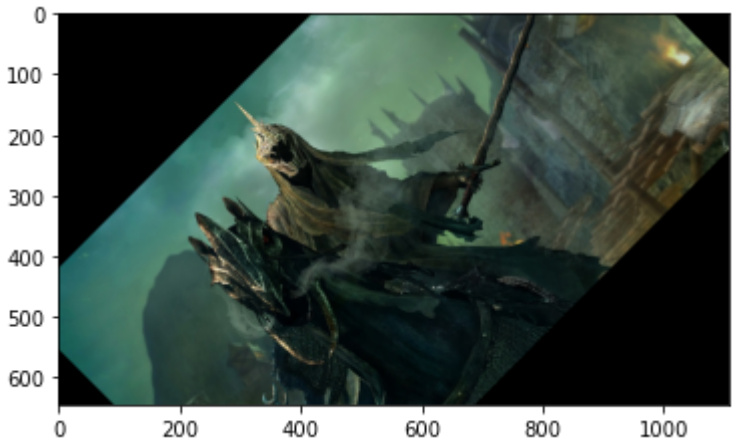
Повернем изображение на 45 градусов:

Ввод [10]:

```
theta = 45
new_image = image.rotate(theta)
```

Ввод [11]:

```
plt.imshow(new_image)
plt.show()
```



Математические операции

Операции с массивами

Вы можете выполнять операции с массивом изображения, используя широковещательную передачу Python, а также добавить константу к значению интенсивности каждого пикселя.

Перед этим вы должны сначала преобразовать изображение PIL в массив numpy.

Ввод [12]:

```
image = np.array(image)
```

Затем добавляем константу в массив изображений:

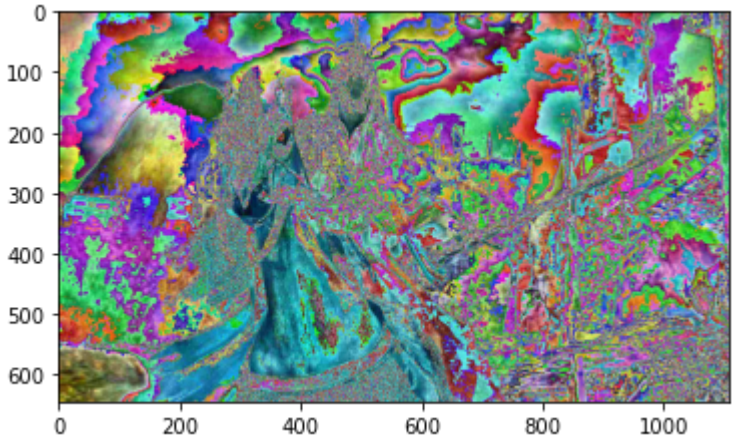
Ввод [14]:

```
new_image = image + 20
plt.imshow(new_image)
plt.show()
```

Вы можете умножить значение интенсивности каждого пикселя на постоянное значение.

Ввод [15]:

```
new_image = 10 * image
plt.imshow(new_image)
plt.show()
```



Вы можете складывать элементы двух массивов одинаковой формы. В этом примере вы генерируете массив случайных шумов с той же формой и типом данных, что и исходное изображение.

Ввод [16]:

```
Noise = np.random.normal(0, 20, (height, width, 3)).astype(np.uint8)
Noise.shape
```

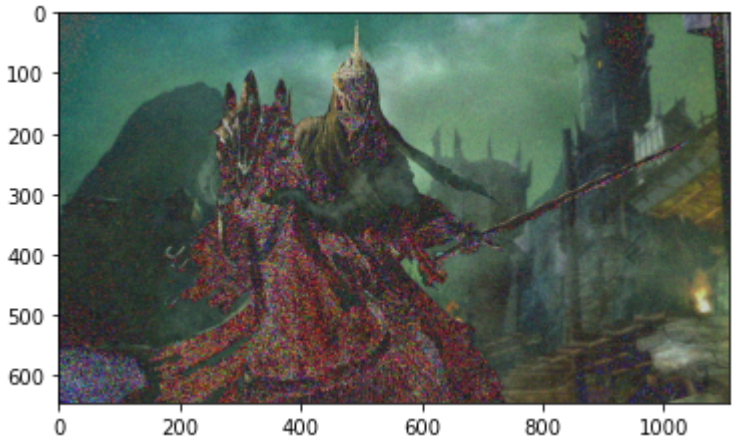
Out[16]:

```
(645, 1109, 3)
```

Далее добавляем сгенерированный шум к изображению и отображаем. Вы видите можете увидеть как испортилось изображение:

Ввод [17]:

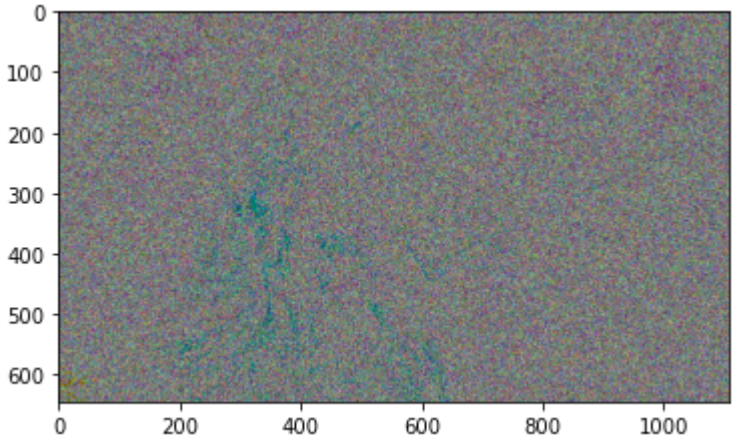
```
new_image = image + Noise

plt.imshow(new_image)
plt.show()
```



При этом вы можете перемножать элементы двух массивов одинаковой формы. Вы можете умножить случайное изображение и первое изображение и отобразить результат.

Ввод [18]:

```
new_image = image*Noise

plt.imshow(new_image)
plt.show()
```



Матричные операции

Изображения в градациях серого представляют собой матрицы. Рассмотрим следующее изображение в оттенках серого:

Ввод [19]:

```
im_gray = Image.open("1.png")
```

Несмотря на то, что изображение серое, оно имеет три канала. Вы можете преобразовать его в одноканальное изображение.

Ввод [20]:

```
from PIL import ImageOps
```

Ввод [21]:

```
im_gray = ImageOps.grayscale(im_gray)
```

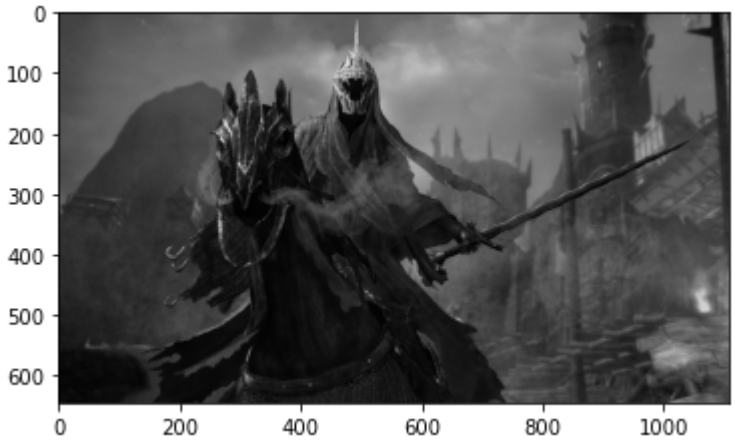
Вы можете преобразовать изображение PIL в массив numpy:

Ввод [22]:

```
im_gray = np.array(im_gray)
```

Ввод [23]:

```
plt.imshow(im_gray,cmap='gray')
plt.show()
```



Вы можете применять алгоритмы, разработанные для матриц. Вы можете использовать разложение по сингулярным значениям, разлагая матрицу изображения на произведение трех матриц.

Ввод [24]:

```
U, s, V = np.linalg.svd(im_gray , full_matrices=True)
```

Можно увидеть, что `s` не прямоугольной формы:

Ввод [25]:

```
s.shape
```

Out[25]:

```
(645,)
```

Вы можете преобразовать `s` в диагональную матрицу `S`:

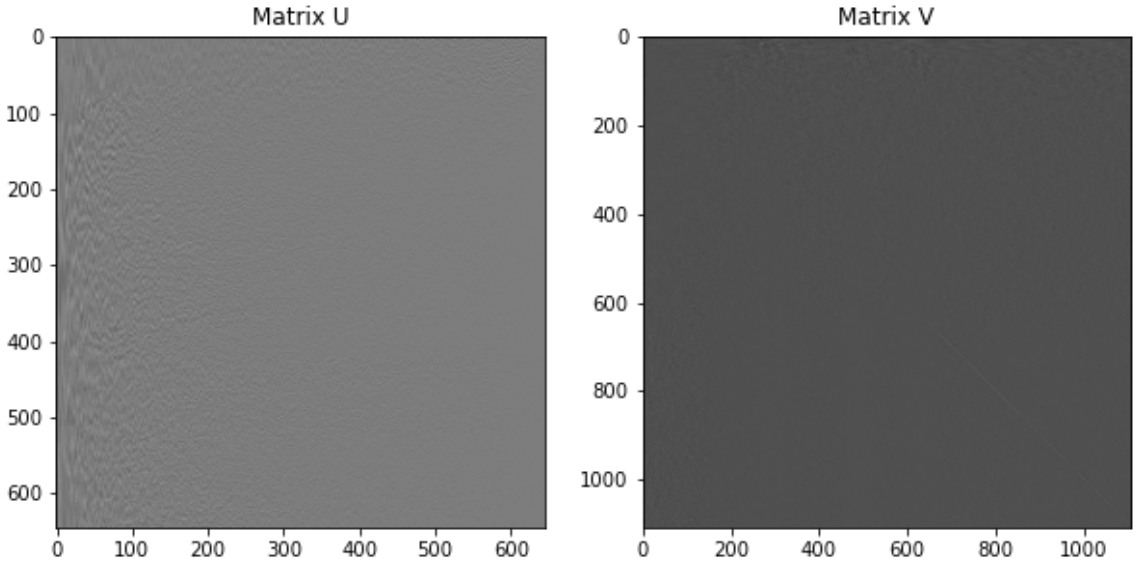
Ввод [26]:

```
S = np.zeros((im_gray.shape[0], im_gray.shape[1]))
S[:image.shape[0], :image.shape[0]] = np.diag(s)
```

Вы можете построить матрицу U и V:

Ввод [27]:

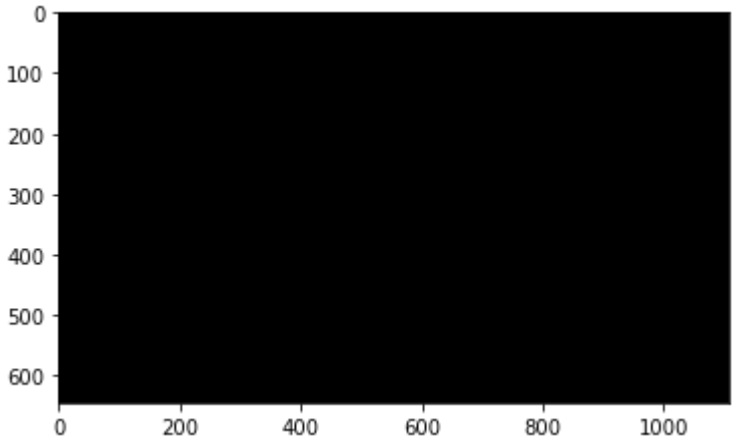
```
plot_image(U, V, title_1="Matrix U", title_2="Matrix V")
```



Мы видим, что большинство элементов в S равны нулю:

Ввод [28]:

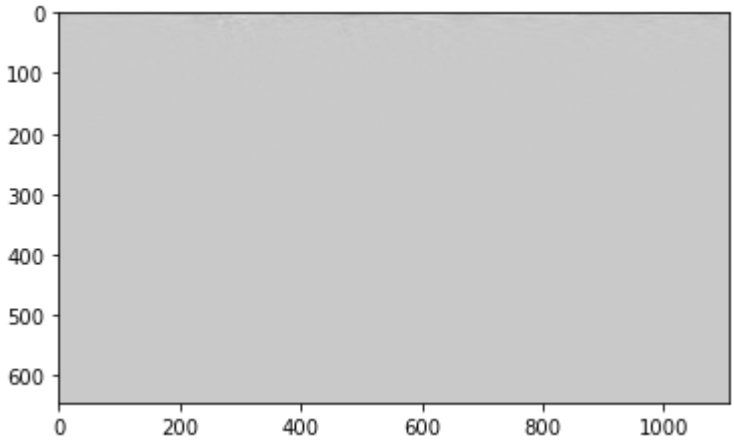
```
plt.imshow(S, cmap='gray')
plt.show()
```



Вы можете найти матричное произведение всех матриц. Во-первых, можно выполнить матричное умножение на `S` и `U`, назначить его на `B` и отобразить результат:

Ввод [29]:

```
B = S.dot(V)
plt.imshow(B,cmap='gray')
plt.show()
```



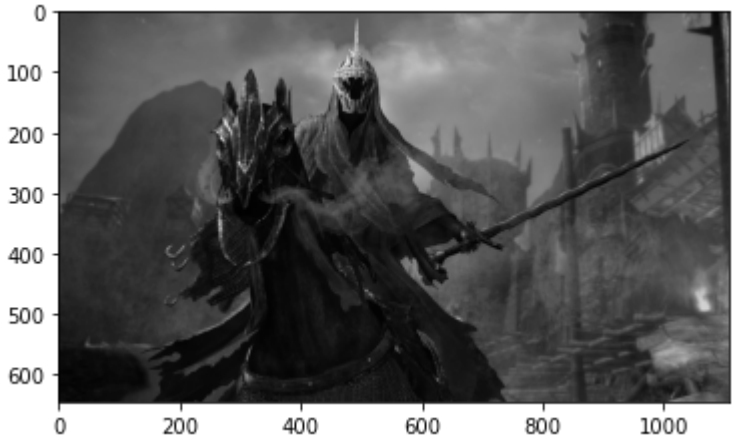
Вы можете найти матричное произведение `U`, `S`, и `B`.Тогда изображение отобразится целиком:

Ввод [30]:

```
A = U.dot(B)
```

Ввод [31]:

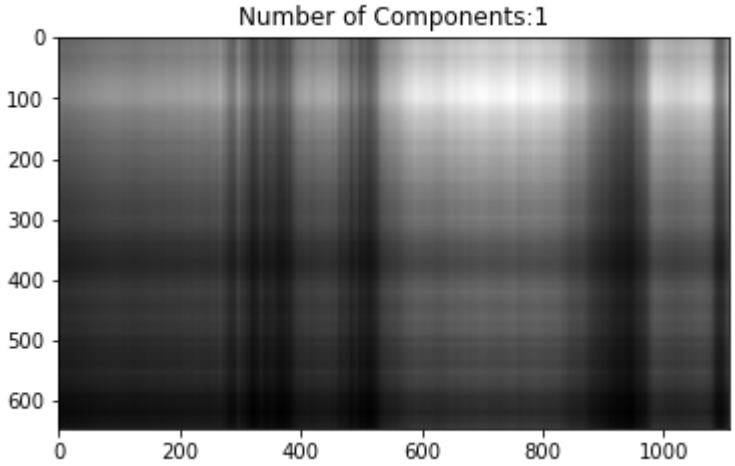
```
plt.imshow(A,cmap='gray')
plt.show()
```

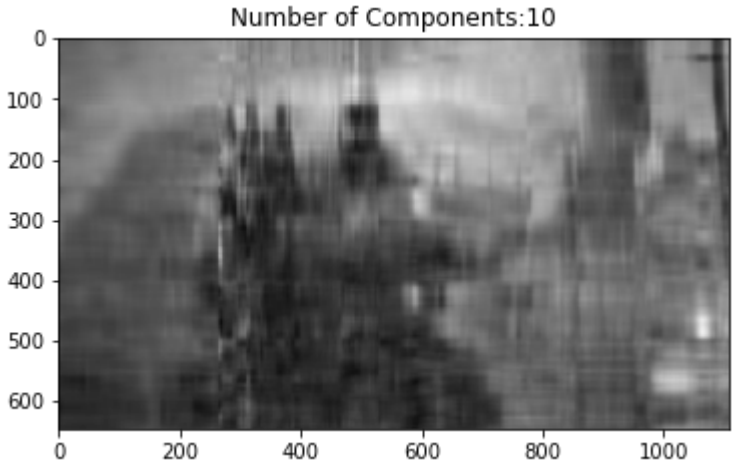


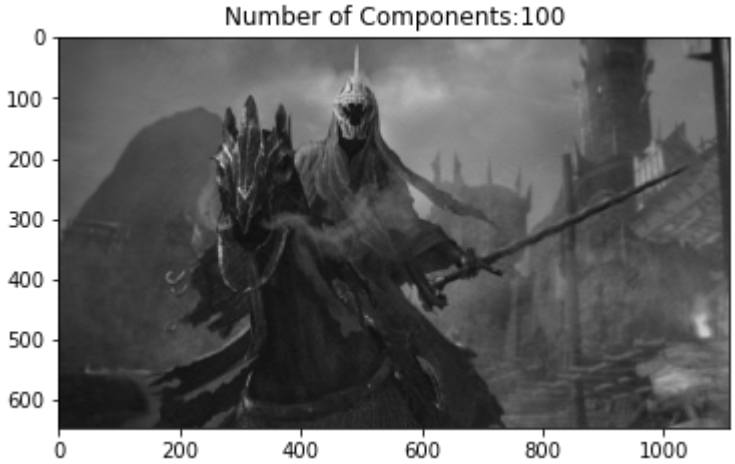
Многие элементы избыточны. Вы можете удалить некоторые строки и столбцы `S` и `V` и приблизить изображение:

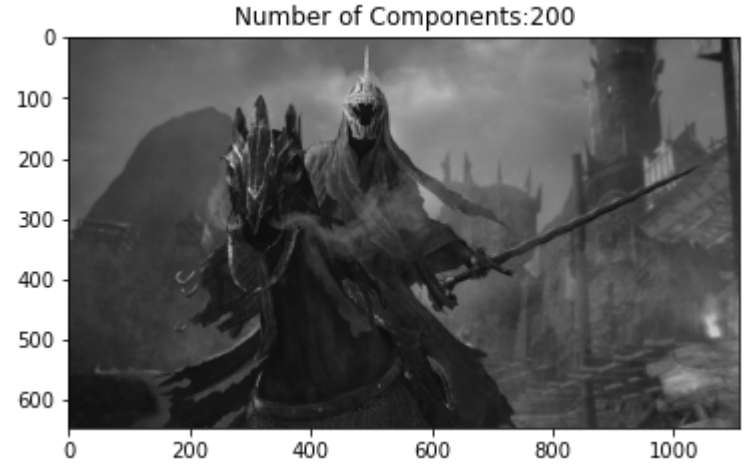
Ввод [32]:

```
for n_component in [1,10,100,200, 500]:
    S_new = S[:, :n_component]
    V_new = V[:n_component, :]
    A = U.dot(S_new.dot(V_new))
    plt.imshow(A,cmap='gray')
    plt.title("Number of Components:"+str(n_component))
    plt.show()
```









Можн понентов для представления изображения в удоб

Пространственные операции при обработке изображений

Пространственные операции используют пиксели в окрестности для определения текущего значения пикселя. Некоторые приложения включают фильтрацию и повышение резкости. Они используются на многих этапах компьютерного зрения, таких как сегментация, и являются ключевым строительным блоком в алгоритмах машинного обучения.

Библиотечки:

Ввод [33]:

```
# Используется для просмотра изображений
import matplotlib.pyplot as plt
# Используется для загрузки изображения
from PIL import Image
# Используется для создания ядер для фильтрации
import numpy as np
```

Ниже предсталена функция, которая построит два изображения рядом.

Ввод [34]:

```
def plot_image(image_1, image_2,title_1="Orignal",title_2="New Image"):
    plt.figure(figsize=(10,10))
    plt.subplot(1, 2, 1)
    plt.imshow(image_1)
    plt.title(title_1)
    plt.subplot(1, 2, 2)
    plt.imshow(image_2)
    plt.title(title_2)
    plt.show()
```

Пространственные операции используют соседние пиксели для определения текущего значения пикселя.

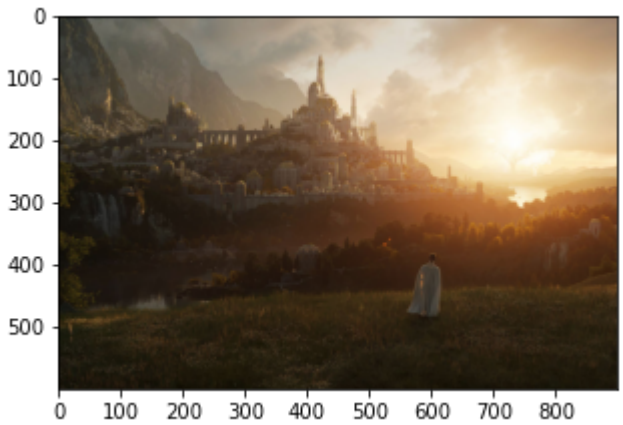
Линейная фильтрация

Фильтрация включает в себя улучшение изображения, например, удаление шума с изображения. Шум может быть вызван плохой камерой или плохим сжатием изображения. Те же факторы, которые вызывают шум, могут привести к нечеткому изображению. Вы можете применять фильтры для повышения резкости этих изображений. Свертка - это стандартный способ фильтрации изображения. Фильтр называется ядром, и разные ядра выполняют разные задачи. Кроме того, свертка используется во многих самых передовых алгоритмах машинного обучения. Вы просто берете скалярное произведение ядра и часть изображения одинакового размера. Затем перекладываем ядро и повторяем.

Рассмотрим следующее изображение:

Ввод [35]:

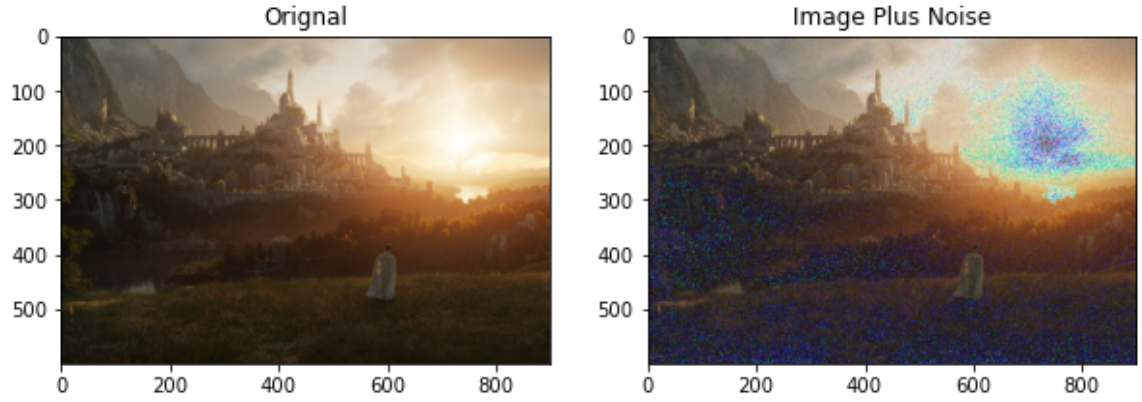
```
# Загружаем изображение из указанного файла
image = Image.open("2.png")
# Отрисовываем изображение
plt.figure(figsize=(5,5))
plt.imshow(image)
plt.show()
```



Изображения, с которыми вы работаете, состоят из значений RGB, которые представляют собой значения от 0 до 255. Ноль означает белый шум, он делает изображение зернистым:

Ввод [36]:

```
# Получаем количество строк и столбцов в изображении
rows, cols = image.size
'''Создаем значения с использованием нормального распределения со средним значением 0
и стандартным отклонением 15, значения преобразуются в unit8, это означает, что
значения находятся в диапазоне от 0 до 255.'''
noise = np.random.normal(0,15,(cols,rows,3)).astype(np.uint8)
# Добавktybt шум к изображению
noisy_image = image + noise
# Создаybtизображение PIL из массива
noisy_image = Image.fromarray(noisy_image)
'''Строим исходное изображение и изображение с шумом,
используя функцию, которую определеили выше'''
plot_image(image, noisy_image, title_1="Orignal", title_2="Image Plus Noise")
```



При добавлении шума к изображению иногда значение может быть больше 255, в данном случае 256. Происходит процедура вычитания, чтобы сохранить его между 0 и 255. Например, рассмотрим изображение со значением RGB, равным 137 и добавим к нему шум со значением RGB, равным 215, чтобы получить значение RGB, равное 352. Затем мы вычитаем 256, общее количество возможных значений от 0 до 255, чтобы получить число от 0 до 255.

Фильтрация шума

Чтобы иметь возможность создавать клиентские ядра и использовать предопределенные фильтры, вы дожете импортировать следующую библиотеку

Ввод [37]:

```
from PIL import ImageFilter
```

Сглаживающие фильтры усредняют пиксели в окрестности, их иногда называют фильтрами нижних частот. Для средней фильтрации ядро просто усредняет ядра в окрестности.

Ввод [38]:

```
'''Создание ядра, которое представляет собой массив 5 на 5,
где каждое значение равно 1/36'''
kernel = np.ones((5,5))/36
# Создание ядра ImageFilter, с указанием размера ядра и сплющенного ядра
kernel_filter = ImageFilter.Kernel((5,5), kernel.flatten())
```

Функция `filter` выполняет свертку между изображением и ядром независимо по каждому цветовому каналу.

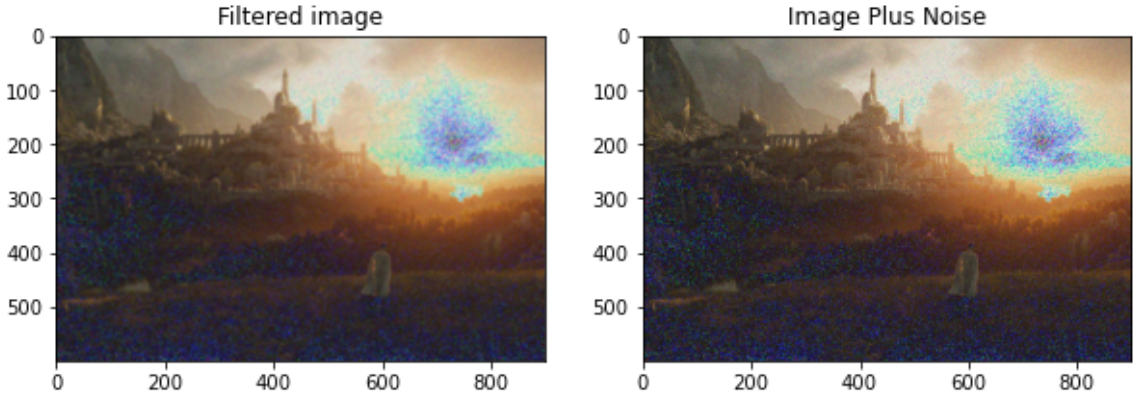
Ввод [39]:

```
# Фильтрация изображения с помощью ядра
image_filtered = noisy_image.filter(kernel_filter)
```

Вы можете построить изображение до и после фильтрации. Вы видите, что шум уменьшился, но изображение получилось размытым:

Ввод [40]:

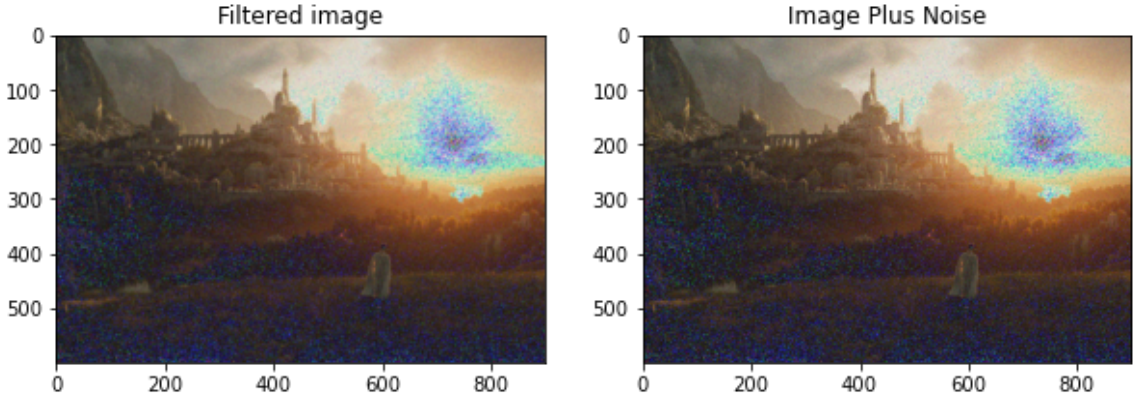
```
'''Отрисовка отфильтрованного изображения и изображения с шумом,
используя функцию, определенную сверху'''
plot_image(image_filtered, noisy_image,title_1="Filtered image",title_2="Image Plus Noise
```



Меньшее ядро сохраняет резкость изображения, но фильтрует меньше шума, ниже вы будете использовать ядро 3x3. Вы можете видеть, что ее линии более резкие на этом изображении, но зеленый шум ярче, чем на отфильтрованном изображении выше.

Ввод [41]:

```
'''Создание ядра, которое представляет собой массив 3 на 3,
где каждое значение равно 1/36'''
kernel = np.ones((3,3))/36
# Создание ядра ImageFilter, с указанием размера ядра и сплющенного ядра
kernel_filter = ImageFilter.Kernel((3,3), kernel.flatten())
# Фильтрация изображения с помощью ядра
image_filtered = noisy_image.filter(kernel_filter)
'''Отрисовка отфильтрованного изображения и изображения с шумом,
используя функцию, определенную сверху'''
plot_image(image_filtered, noisy_image,title_1="Filtered image",title_2="Image Plus Noise
```



Размытие по Гауссу

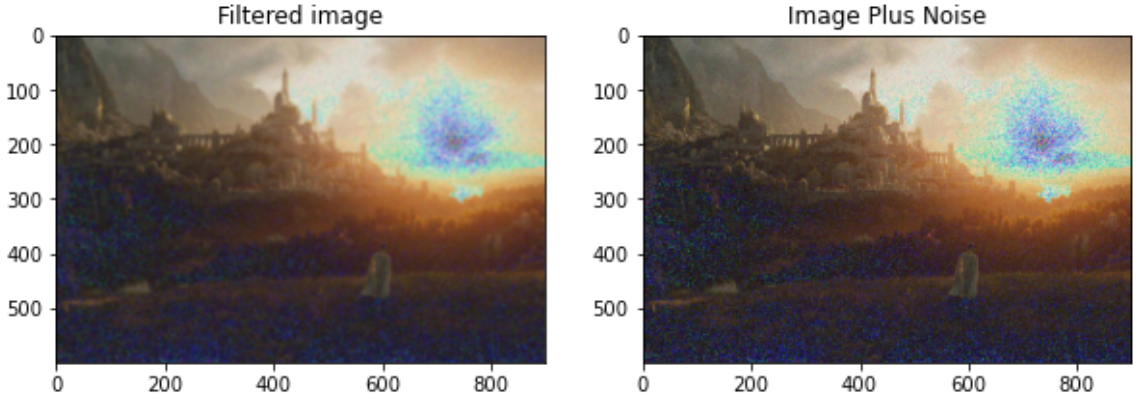
Чтобы выполнить размытие по Гауссу, необходимо применить функцию `filter` к изображению, используя определенный фильтр `ImageFilter.GaussianBlur`

Параметры

`radius` : blur kernel radius, default 2

Ввод [42]:

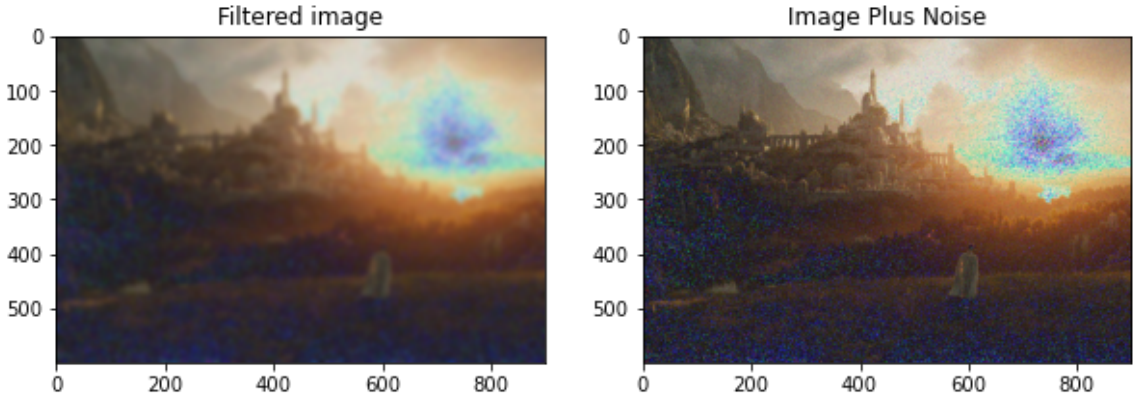
```
# Фильтрация изображения с помощью GaussianBlur
image_filtered = noisy_image.filter(ImageFilter.GaussianBlur)
# Отрисовка отфильтрованного изображения и нефильтрованного изображения с шумом
plot_image(image_filtered , noisy_image,title_1="Filtered image",title_2="Image Plus Nois
```



Пример с использованием ядра 4 на 4

Ввод [43]:

```
'''Фильтрация изображения с помощью GaussianBlur на изображении
с шумом, используя ядро 4 на 4.'''
image_filtered = noisy_image.filter(ImageFilter.GaussianBlur(4))
# Отрисвка отфильтрованного изображения и нефильтрованного изображения с шумом
plot_image(image_filtered , noisy_image,title_1="Filtered image",title_2="Image Plus Nois
```

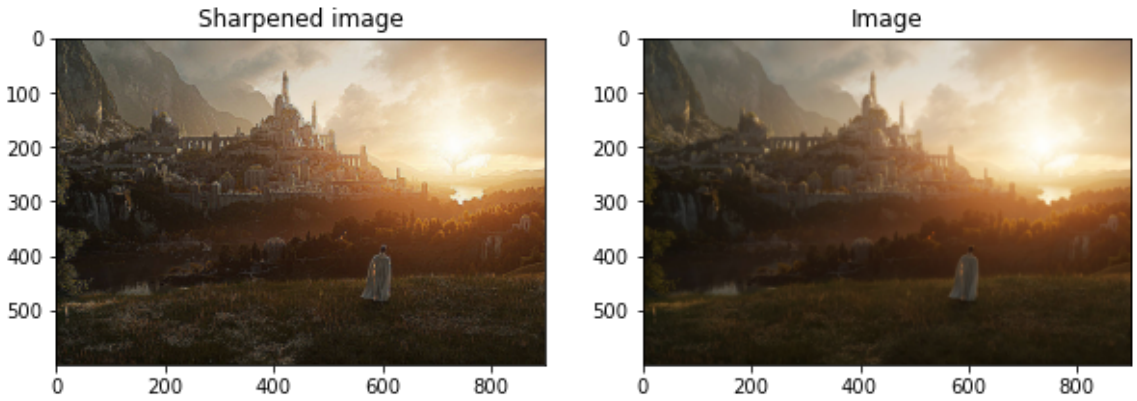


Повышение резкости изображения

Повышение резкости изображения включает в себя сглаживание изображения и вычисление производных. Вы можете добиться повышения резкости изображения, применив следующее ядро.

Ввод [44]:

```
# Общее ядро для повышения резкости изображения
kernel = np.array([[-1,-1,-1],
                   [-1, 9,-1],
                   [-1,-1,-1]])
kernel = ImageFilter.Kernel((3,3), kernel.flatten())
'''Применяем фильтр повышения резкости с использованием ядра
к исходному изображению без шума'''
sharpened = image.filter(kernel)
'''Отрисовка изображения с повышенной резкостью и
исходным изображением без шума'''
plot_image(sharpened, image, title_1="Sharpened image",title_2="Image")
```



Вы также можем повысить резкость с помощью предустановленного фильтра.

Ввод [45]:

```
'''Повышение резкости изображения с помощью предопределенного
фильтра изображения из PIL'''
sharpened = image.filter(ImageFilter.SHARPEN)
''' Отрисовка изображения с повышенной резкостью и
исходным изображением без шума'''
plot_image(sharpened , image, title_1="Sharpened image",title_2="Image")
```

Edges (края)

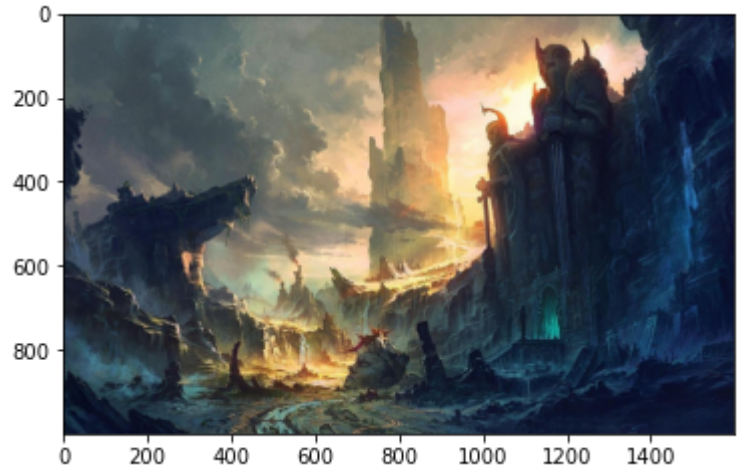
По краям меняется яркость пикселей. В данном случае, градиент функции выводит скорость изменения. Вы можете аппроксимировать градиент изображения в градациях серого с помощью свертки. Рассмотрим следующее изображение:

Ввод [46]:

```
# Загрузка изображения из указанного файла
img_gray = Image.open('3.png')
''' Отрисовываем изображение из массива данных, обратите внимание, что оно двумерное,
а не трехмерное, потому что у него нет цвета.'''
plt.imshow(img_gray ,cmap='gray')
```

Out[46]:

```
<matplotlib.image.AxesImage at 0x252b9e60250>
```



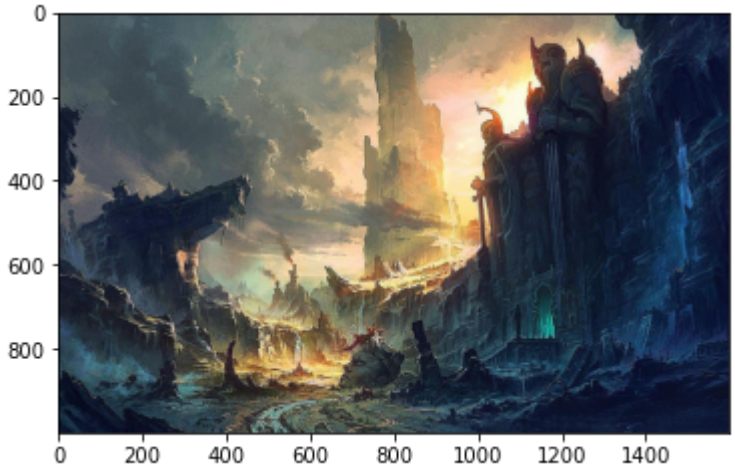
Вы улучшаете края, чтобы они лучше воспринимались, когда вы используете функцию для обнаружения краев.

Ввод [47]:

```
# Фильтрация изображения с помощью фильтра EDGE_ENHANCE
img_gray = img_gray.filter(ImageFilter.EDGE_ENHANCE)
# Отрисовка улучшенного изображения
plt.imshow(img_gray ,cmap='gray')
```

Out[47]:

```
<matplotlib.image.AxesImage at 0x252ba0fc910>
```

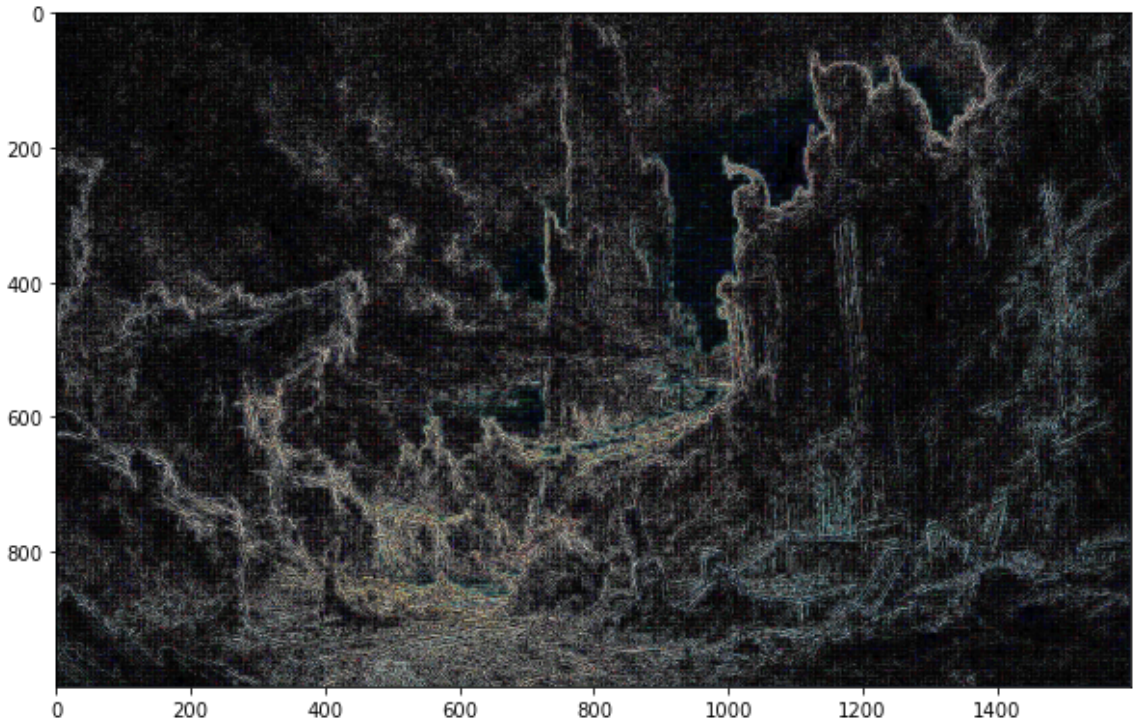


Ввод [48]:

```
# Фильтрация изображения с помощью фильтра FIND_EDGES
img_gray = img_gray.filter(ImageFilter.FIND_EDGES)
# Отрисовка отфильтрованного изображения
plt.figure(figsize=(10,10))
plt.imshow(img_gray ,cmap='gray')
```

Out[48]:

```
<matplotlib.image.AxesImage at 0x252b9dec0d0>
```



Медиана

Медианные фильтры находят медианное значение всех пикселей под областью ядра, и центральный элемент заменяется этим медианным значением.

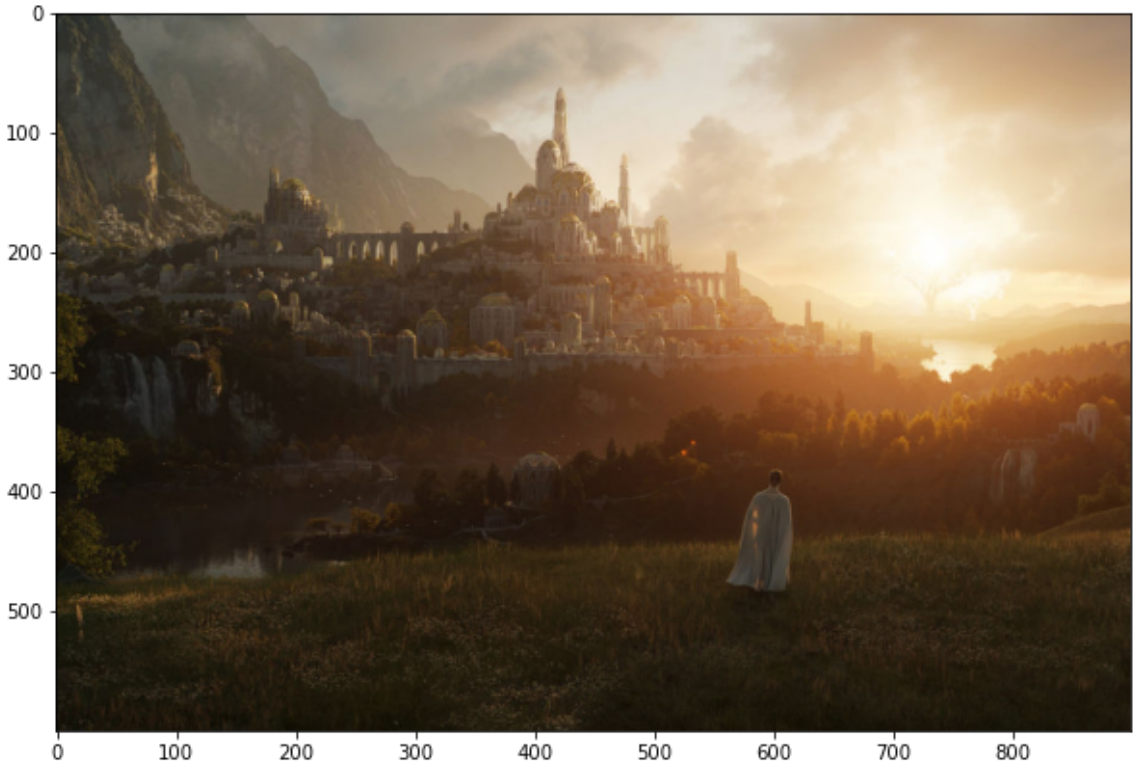
Вы можете применять медианные фильтры к обычным изображениям. Рассмотрим использование медианного фильтра для улучшения сегментации.

Ввод [49]:

```
# Загружаем изображение
image = Image.open("2.png")
# Увеличение изображения при отрисовке
plt.figure(figsize=(10,10))
# Отрисовка изображения
plt.imshow(image,cmap="gray")
```

Out[49]:

```
<matplotlib.image.AxesImage at 0x252b7b57160>
```



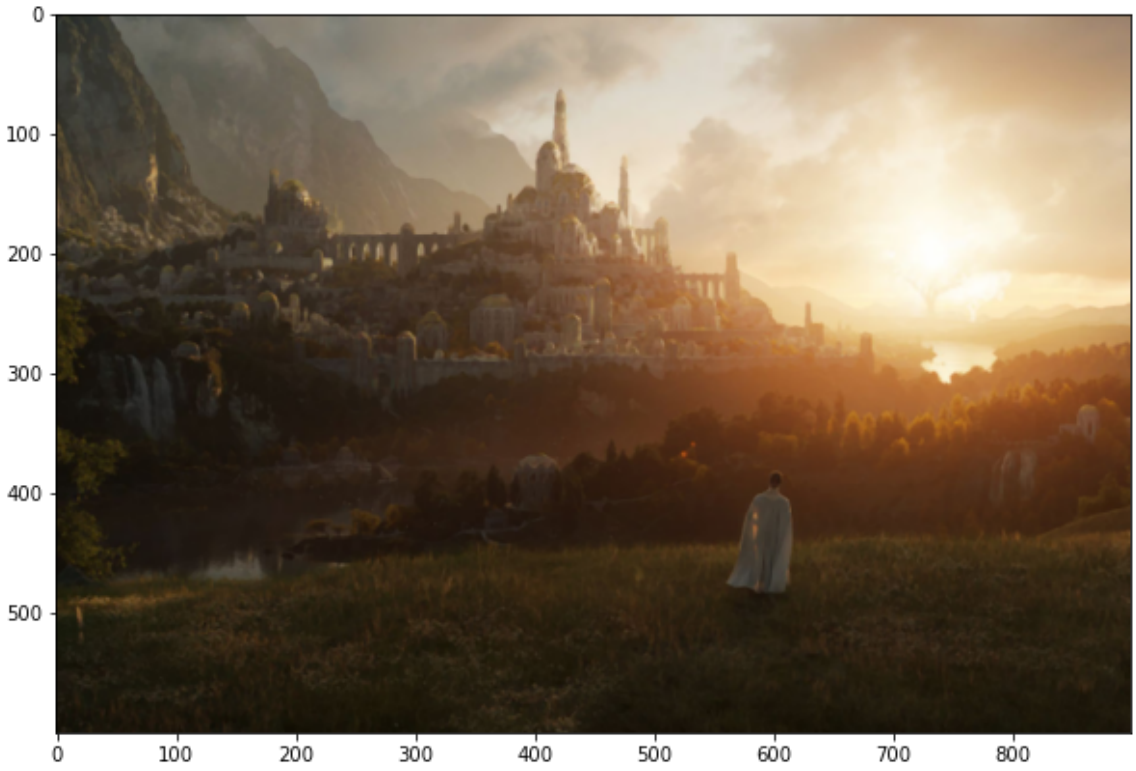
Медианная фильтрация размывает фон, увеличивая сегментацию между объектами и фоном.

Ввод [50]:

```
image = image.filter(ImageFilter.MedianFilter)
plt.figure(figsize=(10,10))
# Отрисовка изображения
plt.imshow(image,cmap="gray")
```

Out[50]:

```
<matplotlib.figure.AxesImage at 0x252b79602e0>
```



Задание:

1. Вам необходимо использовать для обучения свой набор данных, который вы должны собрать сами и загрузить его в колаб через доступ к вашему гугл-диску. Классов должно быть 7, количество экземпляров в классе вариативно.
2. Точность должна быть не меньше 85. Метрику выбираете сами. Метрика должна быть изображена на графике, как в примере из лекции.
3. Чтобы загрузить изображение, вы можете использовать Gradio или просто загружать из консоли.
4. Необходимо отобразить топологию вашей нейронной сети с использование метода model.summary().

Прикрепите файл в формате .ipynb.

Балл за задание: 3.

В этом ноутбуке будет реализована простая модель классификации изображений, с использованием Gradio и Keras. Модель классификации изображений будет классифицировать изображения различных цветов по маркированным классам.

Gradio — это библиотека машинного обучения, которая создает интерактивное приложение для обученной модели машинного обучения. Gradio создает пользовательские интерфейсы (UI), которые позволяют пользователю взаимодействовать с обученной моделью машинного обучения. Она создает веб-интерфейс, который позволяет пользователю тестировать обученную модель и просматривать результаты прогнозирования. Она легко интегрирует пользовательский интерфейс прямо в блокнот Python (либо блокнот Jupyter, либо блокнот Google Colab) без необходимости устанавливать какие-либо зависимости.

Gradio напрямую работает с популярными библиотеками машинного обучения, такими как Sckit-learn, Tensorflow, Keras, PyTorch и Hugging Face Transformers.

Ввод [1]:

```
!pip install gradio
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,)
https://us-python.pkg.dev/colab-wheels/public/simple/ (https://us-pytho
n.pkg.dev/colab-wheels/public/simple/)
Collecting gradio
  Downloading gradio-3.18.0-py3-none-any.whl (14.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 14.2/14.2 MB 30.9 MB/s eta
0:00:00
Collecting aiofiles
  Downloading aiofiles-23.1.0-py3-none-any.whl (14 kB)
Collecting httpx
  Downloading httpx-0.23.3-py3-none-any.whl (71 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 71.5/71.5 KB 6.3 MB/s eta
0:00:00
Requirement already satisfied: fsspec in /usr/local/lib/python3.8/dist-p
ackages (from gradio) (2023.1.0)
Requirement already satisfied: altair>=4.2.0 in /usr/local/lib/python3.
8/dist-packages (from gradio) (4.2.2)
Collecting uvicorn
  Downloading uvicorn-0.20.0-py3-none-any.whl (56 kB)
```

Импорт библиотеки Gradio:

Ввод [2]:

```
import gradio as gr
```

Ипортируем библиотечки:

Ввод [3]:

```
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf

from tensorflow import keras # Импорт keras
from tensorflow.keras import layers # испорт словев keras
```

numpy нужен для преобразования набора данных изображения в массив.

os позволяет использовать функции операционной системы прямо в Google Colab.

Matplotlib - графическая библиотека. Она нужна для визуализации некоторых изображений в Google Colab.

tensorflow необходим для построения модели классификации изображений. Здесь импортируются слои Keras из TensorFlow. Они используются для инициализации сверточной нейронной сети (CNN).

pillow (PIL)используется для обработки и предварительной обработки изображений в наборе данных.

pathlib используется для указания пути к набору данных, чтобы можно было загрузить набор данных изображений в Google Colab.

Путь к набору данных:

Ввод [4]:

```
import pathlib
flowers_dataset_path = "https://storage.googleapis.com/download.tensorflow.org/example_im

# Загрузка набора данных в Google Colab и извлечение

loaded_data = tf.keras.utils.get_file('flower_photos', origin=flowers_dataset_path, untar
loaded_data = pathlib.Path(loaded_data)
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.o
rg/example_images/flower_photos.tgz (https://storage.googleapis.com/downlo
ad.tensorflow.org/example_images/flower_photos.tgz)
228813984/228813984 [==============================] - 4s 0us/step
```

Набор данных изображений имеет пять помеченных классов цветов. Помеченные классы цветов: «тюльпаны», «одуванчик», «маргаритки», «подсолнухи» и «розы». Извлечем розы:

Ввод [5]:

```
roses = list(loaded_data.glob('roses/*'))
print(roses[0]) # отображение выбранного изображения
PIL.Image.open(str(roses[0])) # отображение 1 изображения
```

```
/root/.keras/datasets/flower_photos/roses/4356781875_92c5cd93c0.jpg
```

Out[5]:

Задаем размеры изображения:

Ввод [6]:

```
# задание высоты и ширины изображений, которые будут подаваться в нейронную сеть

set_height,set_width = 180,180 # Входное изображение будет иметь размер 180 на 180 пиксел
batch_size = 32 # размер партии, которая подается на обучение в течение каждой эпохи
```

Настройка тренировочного набора

Необходимо выбрать изображения из набора данных для обучения модели. Сверточная нейронная сеть будет учиться на этих изображениях.

Будем использовать keras.preprocessing для установки тренировочного набора.

Ввод [7]:

```
training_images = tf.keras.preprocessing.image_dataset_from_directory(
  loaded_data,
  subset="training",
  validation_split=0.25,
  seed=123,
  image_size=(set_height, set_width),
  batch_size=batch_size)
```

```
Found 3670 files belonging to 5 classes.
Using 2753 files for training.
```

В приведенном выше выводе: весь набор данных содержит 3670 изображений, а на 2753 изображених будет обучаться CNN.

В функции выше мы используем разделение исходный набор на 0.25, это означает, что на 75% набора данных будет обучаться CNN. Также функция имеет следующие дополнительные параметры:

seed - воспроизведение изображения одинаковых размеров в каждую эпоху.

image_size - размеры изображения (180 * 180), которые мы установили ранее.

batch_size - размер пакета, который мы установили ранее.

Настройка проверочного набора

Ввод [8]:

```
validation_images = tf.keras.preprocessing.image_dataset_from_directory(
  loaded_data,
  subset="validation",
  validation_split=0.25,
  seed=123,
  image_size=(set_height, set_width),
  batch_size=batch_size)
```

```
Found 3670 files belonging to 5 classes.
Using 917 files for validation.
```

В приведенном выше выводе: весь набор данных содержит 3670 изображений, а 917 изображений будут использоваться для оценки точности CNN.

Можно посмотреть все классы:

Ввод [9]:

```
flower_classes = training_images.class_names
print(flower_classes)
```

```
['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
```

Ввод [10]:

```
dataset_classes = 5 # инициализируем классы
```

Импорт последовательной модели

Последовательная модель позволит пользователям создавать несколько слоев сверточной нейронной сети (CNN) друг над другом. Мы будем инициализировать слои Keras слой за слоем и создадим окончательную модель со всеми инициализированными слоями.

Ввод [11]:

```
from tensorflow.keras.models import Sequential

# Инициализируем модель Sequential:

model = Sequential([
  # Нормализация/стандартизация изображения — это процесс создания изображений в наборе д
  # Интенсивность пикселей изображения должна находиться в диапазоне [0, 1].
  layers.experimental.preprocessing.Rescaling(1./255, input_shape=(set_height, set_width,
  # Добавляем слой свертки
  # Conv2D будет двумерным и будет состоять из 16 нейронов.
  # Он имеет 3 канала изображения.
  # Слои имеют отступы same, чтобы нейроны, создающие этот слой, имели одинаковый размер.
  # relu в качестве функции активации, потому что выход этого слоя будет между 0 и положи
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  # Добавляем слой пуллинга
  layers.MaxPooling2D(),
  # Еще слой свертки
  # Этот слой будет иметь 32 нейрона и 3 канала изображения.
  # Он также используется reluв качестве функции активации.
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  # мне лень описывать уже)
  layers.MaxPooling2D(),

  layers.Conv2D(64, 3, padding='same', activation='relu'),

  layers.MaxPooling2D(),
  # Добавляем глаживающий слой
  layers.Flatten(),
  # Также вариативно можно добавить скрытый слой для более точной настройки
  layers.Dense(128, activation='relu'),
  # Выходной слой
  # Этот слой принимает пять инициализированных классов в качестве входных данных.
  # Он используется softmaxв качестве функции активации, поскольку набор данных имеет бол
  layers.Dense(dataset_classes,activation='softmax')
])
```

Сборка CNN

Ввод [12]:

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Функция имеет следующие важные параметры:

optimizer - ставим оптимизатор как Adam. Это улучшит производительность сверточной нейронной сети. Он также обрабатывает ошибки, которые CNN может иметь при обучении.

loss - это функция, которая накапливает все ошибки, с которыми сталкивается CNN во время обучения. Мы используем SparseCategoricalCrossentropy, поскольку набор данных изображения имеет несколько классов (пять классов).

metrics - это функция, которая получит общую оценку точности CNN после обучения. Мы устанавливаем его значение в accuracy.

Ввод [13]:

```
epochs=10 # Задаем количесво эпох
```

Обучение:

Ввод [14]:

```
CNN_model = model.fit(
  training_images,
  validation_data=validation_images,
  epochs=epochs
)
```

```
Epoch 1/10

/usr/local/lib/python3.8/dist-packages/keras/backend.py:5585: UserWarning:
"`sparse_categorical_crossentropy` received `from_logits=True`, but the `o
utput` argument was produced by a Softmax activation and thus does not rep
resent logits. Was this intended?
  output, from_logits = _get_logits(

87/87 [==============================] - 119s 1s/step - loss: 1.2916 - acc
uracy: 0.4646 - val_loss: 1.3266 - val_accuracy: 0.4558
Epoch 2/10
87/87 [==============================] - 105s 1s/step - loss: 0.9784 - acc
uracy: 0.6088 - val_loss: 1.1207 - val_accuracy: 0.5802
Epoch 3/10
87/87 [==============================] - 105s 1s/step - loss: 0.8240 - acc
uracy: 0.6833 - val_loss: 0.9936 - val_accuracy: 0.6041
Epoch 4/10
87/87 [==============================] - 119s 1s/step - loss: 0.7664 - acc
uracy: 0.7120 - val_loss: 0.9975 - val_accuracy: 0.6118
Epoch 5/10
87/87 [==============================] - 116s 1s/step - loss: 0.5060 - acc
uracy: 0.8126 - val_loss: 1.1279 - val_accuracy: 0.6041
Epoch 6/10
87/87 [==============================] - 104s 1s/step - loss: 0.3330 - acc
uracy: 0.8809 - val_loss: 1.1857 - val_accuracy: 0.6292
Epoch 7/10
87/87 [==============================] - 105s 1s/step - loss: 0.2032 - acc
uracy: 0.9368 - val_loss: 1.3444 - val_accuracy: 0.6478
Epoch 8/10
87/87 [==============================] - 115s 1s/step - loss: 0.1173 - acc
uracy: 0.9633 - val_loss: 1.4219 - val_accuracy: 0.6510
Epoch 9/10
87/87 [==============================] - 111s 1s/step - loss: 0.0844 - acc
uracy: 0.9738 - val_loss: 1.3801 - val_accuracy: 0.6478
Epoch 10/10
87/87 [==============================] - 107s 1s/step - loss: 0.0559 - acc
uracy: 0.9873 - val_loss: 1.5858 - val_accuracy: 0.6609
```

Выходные данные показывают CNN loss, accuracy и val_accuracy в течение каждой эпохи. Начальный loss был больше, чем финальный. Это показывает, что ошибки CNN со временем уменьшились.

Начальный accuracy стал меньше финального. Это показывает, что производительность CNN улучшилась по мере увеличения количества эпох. Оценка val_accuracy также увеличивается.

Чтобы использовать CNN для классификации изображений, мы создадим функцию, которая будет принимать входное изображение и возвращать результаты классификации.

Ввод [15]:

```
def predict_input_image(img):
  img_4d=img.reshape(-1,180,180,3)
  prediction=model.predict(img_4d)[0]
  return {flower_classes[i]: float(prediction[i]) for i in range(5)}
```

Функция называется predict_input_image. Чтобы функция работала, входное изображение должно быть четырехмерным. Функция будет использовать img.reshape метод для преобразования входного изображения в четыре измерения. model.predict будет классифицировать входное изображение.

Затем функция возвращает словарь с каждым предсказанным классом и его соответствующей вероятностью. Класс с наибольшей вероятностью будет правильным предсказанием или классификацией.

Затем мы будем использовать Gradio для создания пользовательского интерфейса, который позволит взаимодействовать с обученной CNN.

Реализация Градио

Прежде чем реализовывать пользовательский интерфейс Gradio, необходимо указать размер изображения, которое будет содержать компонент ввода Gradio. Также указывается количество помеченных классов в наборе данных изображения.

Указание размера изображения:

Ввод [16]:

```
image = gr.inputs.Image(shape=(180,180))
```

```
/usr/local/lib/python3.8/dist-packages/gradio/inputs.py:257: UserWarning:
Usage of gradio.inputs is deprecated, and will not be supported in the fut
ure, please import your component from gradio.components
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/gradio/deprecation.py:40: UserWarni
ng: `optional` parameter is deprecated, and it has no effect
  warnings.warn(value)
```

Указание помеченных классов:

Ввод [17]:

```
label = gr.outputs.Label(num_top_classes=5)
```

```
/usr/local/lib/python3.8/dist-packages/gradio/outputs.py:197: UserWarning:
Usage of gradio.outputs is deprecated, and will not be supported in the fu
ture, please import your components from gradio.components
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/gradio/deprecation.py:40: UserWarni
ng: The 'type' parameter has been deprecated. Use the Number component ins
tead.
  warnings.warn(value)
```

Пользователь будет загружать/перетаскивать изображение в пользовательском интерфейсе Gradio, затем обученная CNN классифицирует изображение и выводит классификацию.

Создание и запуск пользовательского интерфейса Gradio:

Ввод [19]:

```
gr.Interface(fn=predict_input_image, inputs=image, outputs=label,interpretation='default'
```

```
Colab notebook detected. This cell will run indefinitely so that you can s
ee errors and logs. To turn off, set debug=False in launch().
Note: opening Chrome Inspector may crash demo inside Colab notebooks.

To create a public link, set `share=True` in `launch()`.

<IPython.core.display.Javascript object>

1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 44ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 33ms/step
Keyboard interruption in main thread... closing server.
```

Out[19]:

Функция gr.Interface создаст пользовательский интерфейс. Он принимает созданную predict_input_image функцию, которая будет классифицировать входное изображение. Он принимает image в качестве входных данных и выводит помеченный класс.

Type *Markdown* and LaTeX: α^2

https://gradio.app/ (https://gradio.app/)