Математические основы компьютерной графики

АЛЕКСАНДР САЯПИН

Математические основы компьютерной графики

Цель курса

Целью данного курса является дать вам представление об основных способах представления информации в компьютере и методах ее обработки.

Обычно обработка изображений не является самоцелью (если вы не фотограф, например), но служит для подготовки изображений к дальнейшей с ними работе. Например, вы можете обрабатывать изображение с камеры компьютера для того, чтобы алгоритмы глубокого обучения лучше выполняли свою работу, например, чтобы камера, являющаяся частью системы автономного управления автомобилем, лучше видела пешеходов на дороге, или чтобы система автоматической сортировки деталей на конвейере лучше их различала.

В курсе мы рассмотрим использование библиотеки компьютерного зрения OpenCV для задач обработки изображений. Она достаточно удобна, широко используется, в том числе и в коммерческих приложениях, и позволит нам использовать основные методы обработки изображений без особых временных затрат. Так же эта библиотека реализована для различных языков программирования и платформ. Мы будем использовать язык Python.

Задачей нашего курса будет разработка последовательности операций, необходимых для подготовки изображений к использованию для анализа с помощью методов машинного обучения.

Вы можете выбрать собственную тему для работы, однако в качестве примера мы будем использовать подготовку изображений египетских иероглифов к их дальнейшему автоматическому распознаванию (например, для статистической обработки - определния того, какие иероглифы встречаются на изображении чаще всего).

Сравните, например, следующие изображения:

Figure 1: ▨▨▨▨ ▨▨▨▨▨

Figure 2: ▨▨▨▨▨▨▨▨▨▨▨ ▨▨▨▨▨▨▨▨▨▨▨

Figure 3: ▨▨▨▨ ▨▨▨▨▨▨

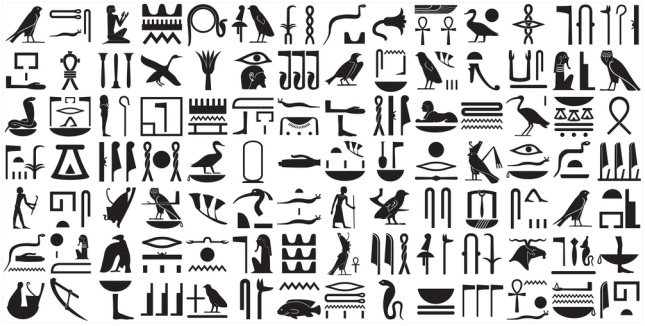

Figure 4: ▨▨▨▨▨ ▨▨▨▨▨▨▨▨▨▨

Figure 5: ▨▨▨▨ ▨▨▨▨▨

Какие из изображений более пригодны для автоматического анализа? Очевидно, что второе и четвертое изображения более пригодня для автоматической обработки. Фактически, это не фотографии - это сгенерированные изображения, на которых нет неравномерной яркости, нет деталей внутри изображений иероглифов (в то время как на фотографиях видны детали обработки камня, которые вовсе не важны для задач распознавания).

Наша цель - разработать последовательность операций над изображением, таких, чтобы иероглифы на изображении могли быть легко выделены автоматически.

Язык Python и необходимые библиотеки

Для работы мы будем использовать широко распространенный язык Python, его библиотеку NumPy и библиотеку компьютерного зрения OpenCV. Библиотека NumPy предназначена для выполнения числовых расчетов среде языка Python, и ориентирована в основном на выполнение операций с матрицами, однако включает в себя так же функции для проведения численных и символьных расчетов, в том числе численного интегрирования, дифференцирования, и других. Для визуализации результатов мы будем использовать пакеты matplotlib и seaborn

Кроме того, для работы мы будем использовать интерактивную web-среду разработки Jupyter Notebook, позволяющую оформлять текст, производить расчеты и строить графики в виде единого интерактивного документа, включающего все вышеперечисленное.

Для установки мы воспользуемся менеджером установки Conda, а точнее, его минимальной версией miniconda. Преимуществом такого подхода является, во-первых, возможность иметь несколько независимых друг от друга сред с разными установленными библиотеками (или разными версиями одних и тех же библиотек), во-вторых, отсутствие проблем с зависимостями при установке библиотек и выборе их версий, в-третьих, возможность легко и без следов удалить ранее установленную среду, если необходимо.

Установка Conda

Для установки miniconda используйте официальный установщик. Скачайте установщик для вашей системы (поддерживаются Windows, MacOS и Linux) и следуйте инструкциям на сайте.

Установка языка Python и необходимых библиотек

Выберите папку на диске, в которой вы хранить ваши работы для данного курса. В этой папке мы установим интетрпретатор языка Python и все необходимые нам библиотеки. После окончания курса, для того, что бы освобождить место на диске, достаточно будет просто удалить папку с работами целиком.

Для установки языка Python и всех необходимых нам библиотек откройте окно терминала в папке, в которой вы планируете выполнять работу. В среде Windows для этого необходимо в проводнике перейти в нужную вам папку, нажать клавишу *Shift* и, удерживая ее, щелкнуть на свободном месте в окне проводника правой кнопкой мыши, а затем выбрать пункт ▯▯▯▯▯▯▯▯ ▯▯▯▯ ▯▯▯▯▯▯▯▯.

В терминале наберите или скопируйте из данного руководства команду

```
conda create --prefix ./mbcg_env python=3 jupyterlab numpy matplotlib opencv
```

Необходимые библиотеки будут скачаны из интернета и установлены на ваш компьютер в подпапку *mbcg_env* в той папке, где вы находитесь. Это займет какое-то время.

Запуск среды Jupyter Notebook

Для начала работы со средой Jupyter Notebook вам необходимо вновь открыть окно терминала в вашей папке с проектами, активировать установленные пакеты командой

```
conda activate ./mbcg_env
```

и запустить среду Jupyter notebook командой

```
jupyter notebook
```

Основы работы с OpenCV и Jupyter Notebook

OpenCV представляет собой библиотеку машинного зрения с открытым исходным кодом, развиваемую сообществом и используемую в том числе и коммерческими фирмами, что говорит о ее зрелости и пригодности для решения самых разных задач.

В библиотеке OpenCV содержится большое количество функций, которые мы будем использовать в работе.

Открытие файлов, просмотр и сохранение изображений

Прежде, чем обрабатывать изображения, необходимо научиться считывать изображения с диска, показывать их на экране компьютера, и сохранять изображения на диск.

Предположим, что все предыдущие шаги вами выполнены, и Jupyter Notebook уже запущен. В браузере вы должны наблюдать следующую картину (рис. 6):

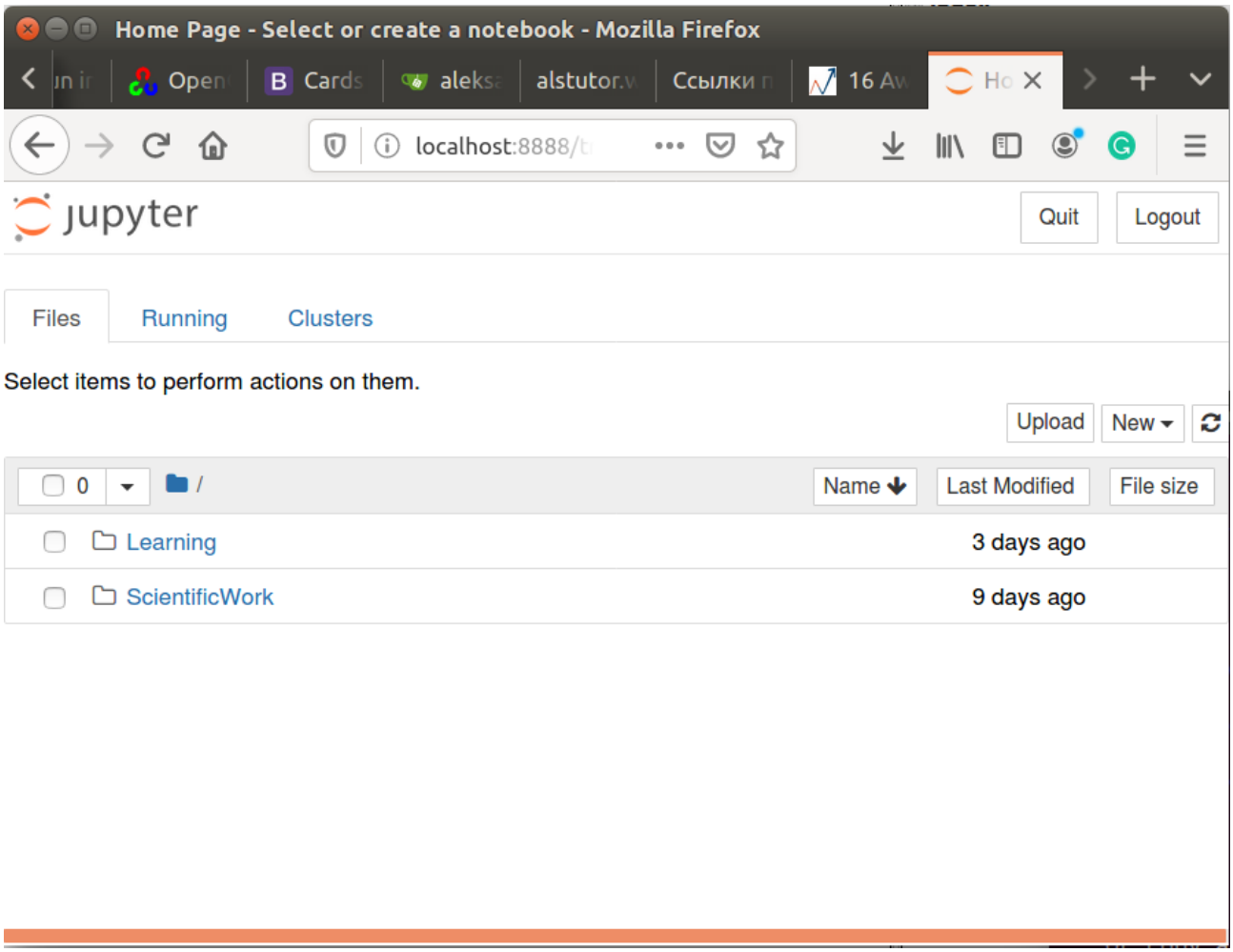

Figure 6: ▯▯▯▯▯▯▯▯▯▯▯ *Jupyter Notebook*

Создайте новый блокнот, нажав на кнопку *New* и выбрав пункт *Python 3* (рис. 7).

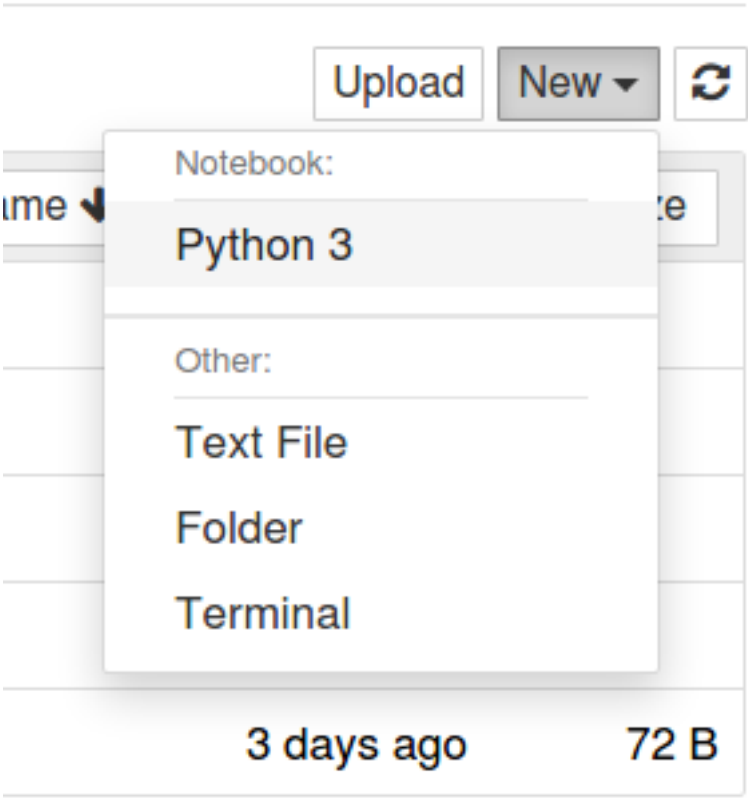

Figure 7: ▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯▯▯

Вы увидите новое окно (рис. 8):

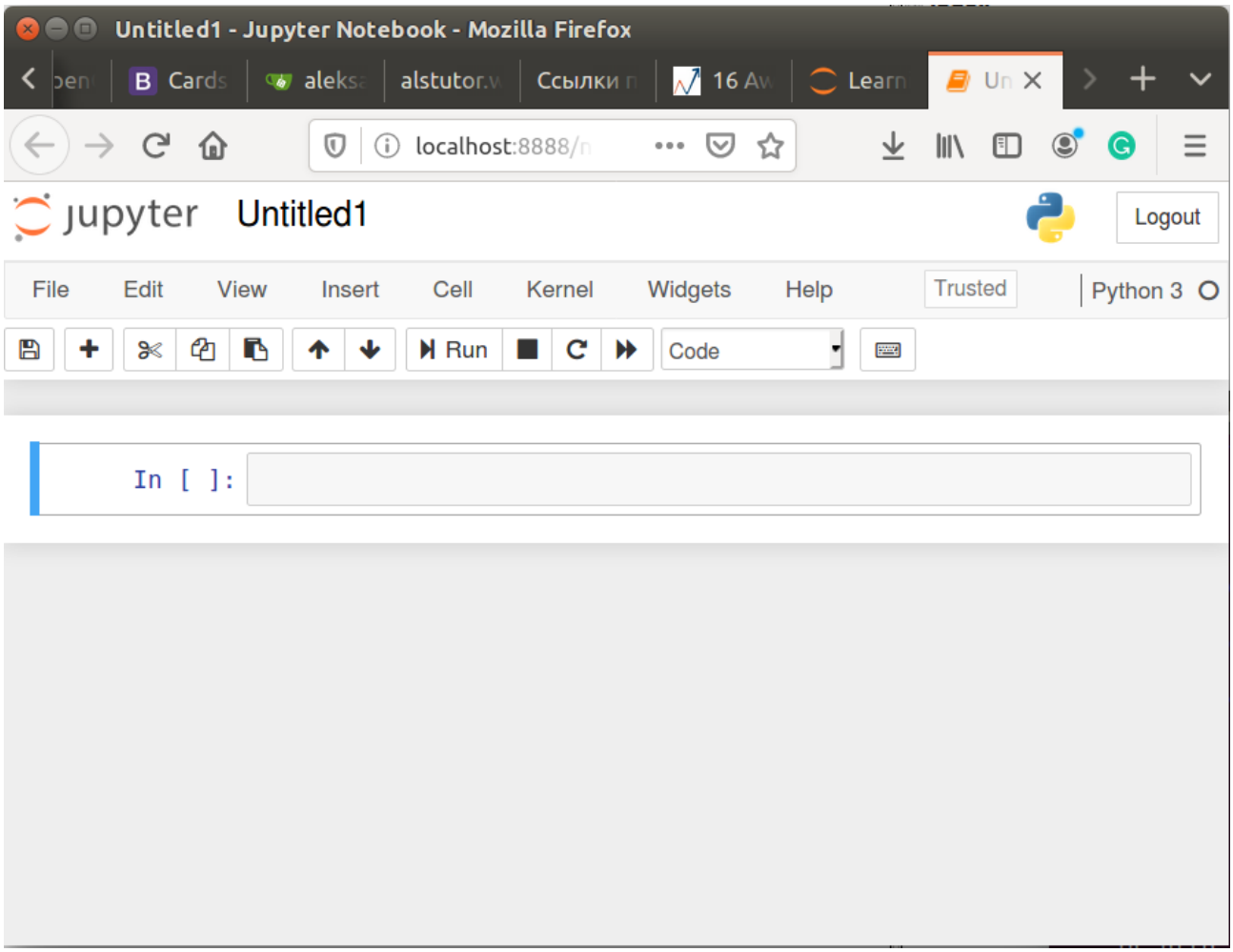

Figure 8: ▯▯▯▯▯▯ ▯▯▯▯▯▯▯▯

Давайте для начала попробуем подключить (или, как говорят профессиональные программисты, импортировать) нужные нам библиотеки.

Введите в блок с надписью *In* (он называется ячейка) следующие строки:

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

и затем нажмите кнопку *Run*.

Если ошибок нет, то никаких сообщений не появится, а ниже возникнет еще одна ячейка с написью *In*.

Что мы только что сделали? Импортировали библиотеку (или, как говорят профессиональные программисты, модуль) под названием *NumPy* (это математическая библиотека, математика нам

понадобится), подключили часть библиотеки *matplotlib* под названием *pyplot*, и указали, что в дальнейшем в нашей программе она будет называться *plt*, импортировали модуль *cv2*, это библиотека компьютерного зрения *OpenCV*, в которой есть многие нужные нам функции.

Последняя команда позволяет просматривать картинки прямо в *Jupyter Notebook*.

Теперь скачайте из интернета любую картинку в формате, и сохраните ее в папку, в которой работаете под именем *index.png*.

Теперь вставьте в новую ячейку следующие команды:

```
image = cv2.imread('index.png')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.imshow(image)
```

Здесь вы видите две команды. Первая, *image = cv2.imread('index.png')* читает картинку из файла *index.png*, и помещает ее в переменную *image*.

Вторая, *image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)*, берет содержимое переменной *image*, и конвертирует цвета из цветовой схемы *BGR*, в цветовую схему *RGB*, помещая результат снова в переменную *image*.

Последняя команда, *plt.imshow(image)*, показывает содержимое переменной *image* в виде картинки на экране.

Нажмите кнопку *Run*.

Вы увидите что-то вроде этого (рис. 4):

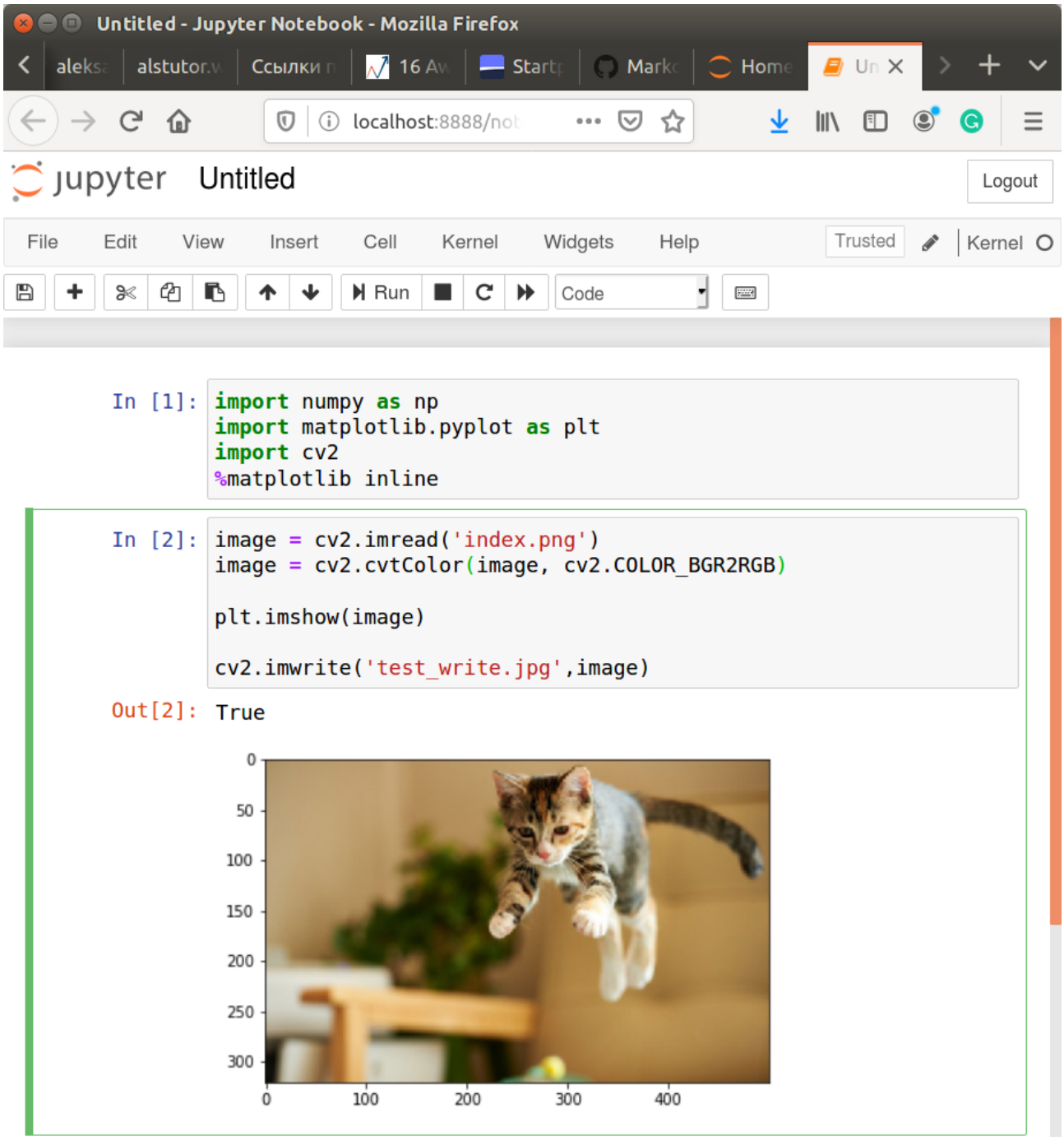

Figure 9: [illegible]

Для того, чтобы просматривать изображение на экране, нам приходится преобразовывать его цвета. Давайте создадим специальную функцию, которая будет показывать изображение на экране.

В новой ячейке введите следующий текст:

```
1 def show_image(image):
```

```
    image_to_show = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    plt.imshow(image)
```

Получение изображения с камеры

Возможно, вам понадобися обработать изображение с веб-камеры, подключенной к компьютеру. Для получения изораженияя с камеры используется следующий код:

```
cam = cv2.VideoCapture(0)
ret, image = cam.read()
```

Первая команда содает объект камеры (в переменной под названием *cam*) в программе, чтобы мы могли его в дальнейшем использовать, а вторая команда читает изображение с камеры, и помещает результаты чтения в две переменные, *ret* и *image*.

В переменной *ret* хранится логическое значение, которое показыевает, было ли чтение успешным. Значение будет *True*, если изображение с камеры было получено успешно, и *False*, если что-то пошло не так, например, камера не подключена или неисправна.

Посмотреть полученное изображение можно так же, как и считанное с диска ранее, командой

```
show_image(image)
```

Введите эту команду и вячейку блокнота и выполните ее, посмотрите результат.

Работа с изображением из сети интернет

В новой ячейке введите следующие команды:

```
import urllib

url = 'https://s14-eu5.startpage.com/cgi-bin/serveimage?url=https%3A%2F%2
    Fwww.cleverfiles.com%2Fhowto%2Fwp-content%2Fuploads%2F2018%2F03%2Fminion
    .jpg&sp=ecbd567c848e2d7eb092cfa9921acb37&anticache=847987'
resp = urllib.request.urlopen(url)
image = np.asarray(bytearray(resp.read()), dtype="uint8")
image = cv2.imdecode(image, cv2.IMREAD_COLOR)
image = cv2.resize(image,(160, 120))
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.imshow(image)
```

Первой строкой мы импортируем модуль для работы с сетевыми сервисами (скачиванием из локальной сети и из сети интернет), затем задаем в переменной *url* адрес изображения в сети,

которе хотим скачать, создаем переменную *cam*, которая соответствует камере, подключенной к роботу, а затем получаем кадр с камеры так же, как раньше получали кадр с камеры, подключенной к компьютеру.

Последней командой мы вызываем нашу функцию, которая показывает изображение на экране.

Представление и параметры растровых изображений

Для представления графической информации в компьютере обычно используются два основных подхода: растровый и векторный. Векторный в нашем курсе не рассматривается, однако базовое представление о ней вы можете получить в соответствующей статье википедии.

В нашем курсе мы сосредоточимся на растровой графике. Базовые представления о сути растровой графики можно получить из соответствующей статьи википедии.

Важнейшим понятием, используемым при работе с растровыми изображениями является пиксел.

Важной характеристикой растрового изображения, тесно связанной с понятием [illegible], является [illegible].

Разрешение

Поскольку растровое изображение фактически представляет собой прямоугольную матрицу чисел (три матрицы для цветных изображений), разрешением изображения называется размер этой матрицы. Чем больше разрешение, тем выше качество изображения, но и тем больший объем памяти занимает изображение в памяти компьютера или на диске.

Операции масштабирования изображений широко используются при работе с фоографиями (и другими типами изображений). Следует помнить, что увеличение изображения обычно мало востребовано, так как (в отличие от того, как это изображается в фильмах), увеличение разрешения изображения не приводит к появлению на нем новых деталей. То есть, если у вас есть фотография группы людей с невысоким разрешением, и вы увеличиваете разрешение в надежде различить лицо человека, у вас ничего не получится - лицо будет размытым и новые детали не появятся. Так происходит потому, что в исходном изображении, например, нос занимал 3 пикселя, псоле операции увеличения разрешения в два раза он будет занимать 6 пикселей, но информации о цвете новых 3 пикселей, которые были добавлены при увеличении разрешения, в исходном изображении не было. Таким образом, алгоритм увеличения изображения должен был “придумать” эту информацию.

Масштабирование изображений

За масштабирование изображений в библиотеке *OpenCV* отвечает функция *resize*.

В свободной ячейке блокнота введите команды

```
image = cv2.resize(image,(80, 60))
show_image(image)
```

Здесь мы выполнили масштабирование изображения, хранящегося в переменной image, задав ему новое разрешение 80x60, поместили новое изображение в ту же переменную *image*, и вывели изображение на экран, используя функцию *show_image*, которую мы добавили ранее.

У функции *resize* есть так же дополнительный параметр, указывающий, какой алгоритм масштабирования будет использован.

Давайте попробуем масштабировать новое, уменьшенное изображение, до исходного размера, используя различные алгоритмы масштабирования.

Например, введите в свободную ячейку следующие команды:

```
scaled_up_image = cv2.resize(image,(640, 480), cv2.INTER_NEAREST)
show_image(image)
```

запустите ячейку на выполнение и оцените результат.

Примеры использования функции *resize* описаны в статье на наглийском языке.

Задание для работы

1. Откройте файл изображения *egypt_luxor_karnak_hieroglyphics_wall_font_characters-1160772.jpg!d.jpeg*.
2. Выполните увеличение уменьшенного изображения иероглифов с использованием различных методов интерполяции до размера 1800x2252.
3. Опишите эти методы и опишите область их применимости.
4. Опишите, какие из методов могут быть использованы для уменьшения изображений.
5. Опишите, какие из методов могут быть рекомендованы для увеличения фотографий иероглифов с целью их последующего анализа.
6. Сохраните увеличенное изображение

Аффинные преобразования

Очевидно, что идельное расположение камеры для съемки интересующего нас объекта - это перпедикулярно поверхности фотографируемого объекта. Таким образом мы получаем наименьшие геометрические искажения (конечно, при условии, что объект, который мы фотографируем - плоский).

Однако в реальности такие условия съемки доступны не всегда. Что же делать, если мы проводили съемку не под прямым углом к поверхности, и получили искажения?

Нам помогут геомерические преобразования изображения, в частности, аффинное преобразование.

Подробнее с теорией аффинных преобразований можно познакомиться в википедии и викиучебнике. Особенностью аффинных преобразований является то, что прямые, параллельные в оригинальном изображении, останутся параллельными в преобразованном.

С практической точки зрения аффинные преобразования в OpenCV выполняются следующим образом:

1. Построить треугольник в координатах исходного изображения. Обычно в качестве вершин треугольника выбираются какие-то характерные точки изображения, например, если в изображении присутствует какая-то рамка, логично в качестве двух из трех вершин треугольника выбрать два угла рамки, а в качестве третьей вершины - середину противолежащей стороны.
2. Построить треугольник в координатах конечного изображения, с точками, заданными в той же последовательности. При этом, если в качестве вершин треугольника мы выбрали углы рамки, присутствующей на исходном изображении, то в конечном изображении они станут углами изображения, а третья точка - серединой противолежащей стороны.
3. Построить аматрицу аффинного преобразования.
4. Применить полученную матрицу к исходному изображению.

Расмотрим пример: в блокноте, который мы уже создали в предыдущей работе, добавим новую ячейку:

```
image = cv2.imread('181888_big.jpg')
image.shape
```

В следующей ячейке задаим координаты вершин треугольников (в исходном изображении и в преобразованном изображении соответственно):

```
src_tri = np.array( [[75, 75], [200, 320], [650, 200]] ).astype(np.float32)
dst_tri = np.array( [[0, 0], [0, 423], [753, 211]] ).astype(np.float32)
```

Точки на изображении можно видеть вот здесь:

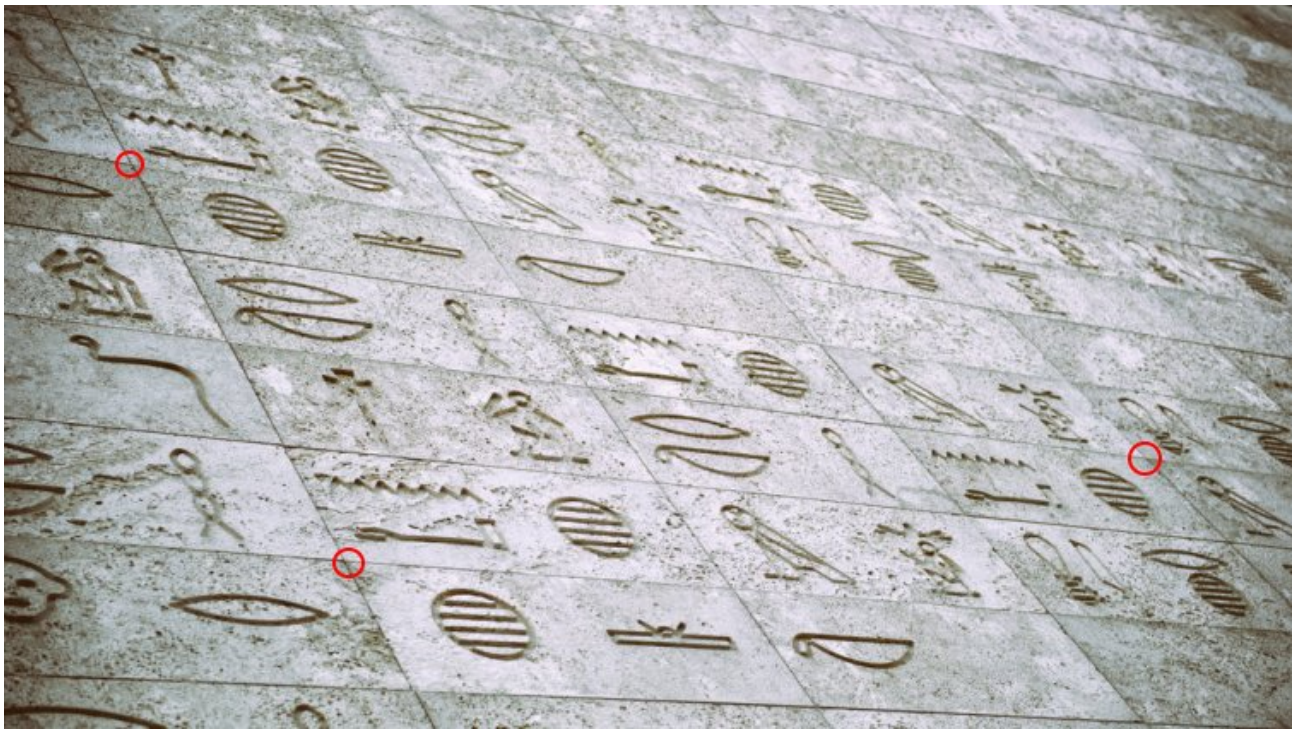

Figure 10: *Source triagle*

Далее найдем матрицу преобразования, выполним преобразование и покажем результирующее изображение:

```
warp_mat = cv2.getAffineTransform(src_tri, dst_tri)
dst_image = cv2.warpAffine(image, warp_mat, (753, 423))
plt.imshow(dst_image)
```

В результате получим следующее изображение

Figure 11: *Affine Translation Result*

Как можно видеть, изображение транформировалось в нужном нам направлении, однако результат пока не идеален. Это произошло потому, что мы изменили положение двух границ прямоугольника, и остались еще две противоположные. Второе преобразование вам необходимо выполнить самостоятельно.

Задание для работы

1. Откройте увеличенный в первом задании файл.
2. Добейтесь того, чтобы плита занимала все изображение, а ее стороны были параллельны сторонам изображения.

Перспективные преобразования

При обработке снимков иногда необходимо исправлять перспективные искажения, например, преобразовать непараллельные изображения исходного изображения в параллельные. Для этого используются перспективные преобразования.

Перспективные преобразования в OpenCV выполняются следующим образом:

1. Построить прямоугольник в координатах исходного изображения, выбирая какие ключевые точки изображения в качестве вершин прямоугольника.Прямоугольник, вероятнее всего, будет неправильным.

2. Построить тпрямоугольник в координатах конечного изображения, с точками, заданными в той же последовательности. Если вы выбрали в качестве ключевых точек вершины какого-либо прямоугольника на исходном изображении, в конечном изображении они станут координатами углов изображения.
3. Построить матрицу перспективного преобразования.
4. Применить полученную матрицу к исходному изображению.

Вновь рассмотрим изображение, которе мы использовали ранее

```
image = cv2.imread('181888_big.jpg')
```

В следующей ячейке задаим координаты вершин прямоугольников (в исходном изображении и в преобразованном изображении соответственно):

```
pts1 = np.float32([[76,94],[239,402],[440,157],[664,419]])
pts2 = np.float32([[0,0],[0,422],[752,0],[752,422]])
```

Далее нам необходимо, как и в предыдущем случае, построить матрицу преобразований и применить ее к изображению:

```
M = cv2.getPerspectiveTransform(pts1,pts2)
dst = cv2.warpPerspective(image,M,(752,422))
plt.imshow(dst)
```

В результате получим следующее изображение

Figure 12: *Perspective Translation Result*

Задание для работы

1. Откройте файл *egypt_luxor_karnak_hieroglyphics_wall_font_characters-1160772.jpg!d.jpeg*.
2. Добейтесь того, чтобы плита занимала все изображение, а ее стороны были параллельны сторонам изображения.

Представление цвета и системы координат

Модель RGB

Глубина цвета

Модель HSV

Простейшие методы обработки изображений

Яркость и контраст

Масштабирование изображения

Преобразование поворота

Цифровые фильтры изображений

Компьютерная геометрия

Двумерные преобразования

Однородные координаты

Двумерное вращение вокруг произвольной оси

Трехмерные преобразования и проекции

Проекции