Нижегородский государственный университет им. Н.И. Лобачевского
Институт информационных технологий, математики и механики
Кафедра Математического обеспечения и суперкомпьютерных технологий

Летняя школа Intel – ННГУ по компьютерному зрению

Практическая работа №1
Инструменты разработки ПО и знакомство с OpenCV

Васильев Е.П.

Нижний Новгород
2021

1 Цели и задачи работы

Цель работы - освоить следующие инструменты разработки программного обеспечения:

- Система контроля версий Git
- Библиотека OpenCV

Основные задачи

1. Создать копию репозитория к себе в аккаунт Github и клонировать на локальную машину
2. Разработать:
    1. приложение для открытия исходного изображения из файла и сохранения результирующего изображения в файл;
    2. фильтр для перевода изображения в оттенки серого;
    3. фильтр для изменения размера изображения.
3. Сделать "commit"(зафиксировать) разработанные методы в локальном репозитории.
4. Отправить "pull request" в основной репозиторий, содержащий шаблоны практических заданий школы.
5. Выполнить дополнительные задания.

2 Общая последовательность действий

1. Установить Python и OpenCV
2. Сделать форк upstream-репозитория.
3. Клонировать origin-репозиторий к себе на локальную.
4. Создать рабочую ветку.
5. Реализовать отображение картинки на экране при помощи OpenCV. По мере готовности не забывайте выкладывать изменения в рабочую ветку на сервер.
6. Реализовать фильтр, который будет переводить изображение в оттенки серого.
7. Реализовать фильтр, который будет изменять размер изображения.
8. Реализовать детектирование с помощью детектора Хаара.
9. Реализовать сохранение изображения.
10. Сделать Pull Request в upstream-репозиторий.

3 Настройка рабочей среды

3.1 Установка Python 3

Для работы лучше всего использовать последнюю версию Python 3.8, которую можно скачать с официального сайта [4].

Информация, приведенная ниже, актуальна для пользователей операционной системы Windows. Если при установке путь до бинарных файлов Python не добавлен в переменную окружения **`PATH`**, то для работы с Python из командной строки требуется выполнить следующую команду, чтобы сделать его доступным из командной строки:

```
set PATH="C:\Program Files (x86)\Python3.8\bin;%PATH%"
```

3.2 Создание виртуальной среды Python

Библиотеки на языке Python устанавливаются в систему подобно тому, как устанавливаются программы в операционную систему. При этом может потребоваться несколько разных версий одной библиотеки. Для этого можно создать несколько окружений – ***виртуальных сред Python***, в которых будут установлены разные версии библиотеки.

Для того, чтобы создать новую виртуальную среду Python, выполните команды, показанные ниже.

```
mkdir openvino-virtual-environments && cd openvino-virtual-environments
python -m venv openvinoenv
```

Чтобы активировать виртуальную среду, выполните одну из следующих команд.

В Windows:

```
openvino-virtual-environments\bin\activate.bat
```

В Linux:

```
source openvino-virtual-environments/bin/activate
```

В дальнейшем при работе с Python необходимо активировать существующую виртуальную среду Python. Для удобства можно сохранить все команды в текстовый файл, чтобы в следующий раз не набирать их вручную, либо создать скрипт, активирующий Python, виртуальную среду и OpenVINO.

3.3 Установка OpenCV

Самостоятельная сборка OpenCV может занять длительное время, поэтому лучше устанавливать уже собранную версию.

```
python -m pip install -U pip
pip install opencv-python
```

В системе пакетов conda пакет называется не opencv-python, а просто opencv.

3.4 Работа с Git / Github

В данном разделе описана типичная последовательность действий, которую необходимо выполнить перед тем, как начать работать с проектом. Далее для определенности используется репозиторий `CV-SUMMER-CAMP-2021`.

1. Создать аккаунт на github.com, если такой отсутствует. Для определенности обозначим аккаунт `github-account`.
2. Сделать fork репозитория https://github.com/itlab-vision/CV-SUMMER-CAMP-2022 (в терминологии Git upstream-репозиторий) к себе в личный профиль с названием `github-account`. В результате будет создана копия репозитория https://github.com/itlab-vision/CV-SUMMER-CAMP-2022 (origin-репозиторий).
3. Клонировать origin репозиторий к себе на локальный компьютер, воспользовавшись следующей командой:

```
git clone https://github.com/   <github-account>   /CV-SUMMER-CAMP-2022
```

4. Перейти в директорию `CV-SUMMER-CAMP`:

```
cd ./CV-SUMMER-CAMP-2022
```

5. Настроить адрес upstream-репозитория (потребуется при обновлении локальной версии репозитория):

```
git remote add upstream https://github.com/itlab-vision/CV-SUMMER-CAMP-2022
```

6. Настроить имя пользователя и e-mail, из под которого будут выполняться все операции с репозиторием Git:

```
git config --local user.name "github-account"
git config --local user.email "github-email"
```

Примечание: если не выполнить указанную операцию при попытке размещения изменений на сервер, они попадут под аккаунтом пользователя компьютера.

7. Настроить редактор, который будет использован, если вносятся изменения в историю репозитория (в частности, при слиянии веток).

```
git config --local core.editor "'C:/Program Files (x86)/Notepad++/notepad++.exe' -n -w"
```

Когда сделан форк репозитория у вас создается по умолчанию единственная ветка master. Тем не менее, при решении независимых задач следует создавать рабочие ветки. Далее показаны основные команды для управления ветками на примере ветки `practice-1`.

8. Получить список веток:

```
git branch [-v]
      # [-v] - список с информацией о последних коммитах
```

9. Создать ветку:

```
git branch practice-1
```

10. Создать ветку `practice-1` и перейти в нее:

```
git checkout [-b] practice-1
      # [-b] - создание и переход в ветку <branch_name>
```

11. Удалить ветку в локальном репозитории:

```
git branch -d <branch_name>
```

12. Удалить ветку на сервере:

```
git push [remotename] :[branch]
      # [remotename] - имя удалённого репозитория. Если следовать приведённой
      #     инструкции, то origin - репозиторий пользователя github-account,
      upstream -
      #     репозиторий itlab-vision/CV-SUMMER-CAMP
```

При работе с файлами в ветке необходимо управлять изменениями. Далее приведен перечень основных команд в предположении, что текущей рабочей веткой является `practice-1`.

1. Получить список текущих изменений:

```
git status
```

2. Пометить файл как добавленный в текущую ветку репозитория (файл будет добавлен после выполнения команды `commit`):

```
git add [<file_name>]
      # <file_name> - название файла для добавления в commit
      #     если вместо имени указан символ *, то будут добавлены все новые
      файлы,
      #     не совпадающие с масками, указанными в .gitignore
```

3. Добавить изменения в текущую ветку локального репозитория:

```
git commit [-m "<message_to_commit>"] [-a]
      # [-a] - автоматически добавляет изменения для существующих на сервере
      файлов
      #     без выполнения команды git add
      # [--amend] - перезаписывает последний коммит (используется, если не
      забыты
      #     изменения)
```

4. Разместить изменения, которые были добавлены в локальный репозиторий с помощью команды `commit`:

```
git push [-u] origin [practice-1]
      # [-u] - отслеживать версию ветки [practice-1] на удалённом сервере
      #     (origin). Позволяет получать изменения с сервера при помощи
      команды git pull
      #     без явного указания имени удалённого репозитория и имени ветки.
```

5. Получить изменения с сервера при помощи команды `pull` и слить их с отслеживаемыми ветками:

```
git pull [remotename [<branch name>]]
```

6. Удалить файлы или директории (!без опции `-f` для файлов, состояния которых совпадают с состояниям на сервере):

```
git rm [-f] [--cached]
      # [-f] - принудительное удаление (файла с измененным состоянием)
      # [--cached] - удаление файлов на сервере, но не в локальной директории
```

7. Переименовать файлы (или 3 команды: `mv`, `git rm`, `git add`):

```
git mv <file_from> <file_to>
```

Когда в проекте работает несколько человек, то вполне естественная ситуация - необходимость слияния изменений и разрешение конфликтов.

1. Слияние (вариант 1):

```
git merge upstream/master # слияние изменений из ветки upstream в master
git merge master # слияние изменений из ветки master в текущую ветку
```

2. Слияние (вариант 2):

```
git checkout <branch_name> # переход в ветку <branch_name> (при
необходимости)
git rebase <base_branch> [<branch_name>]
# слияние изменений из ветки <base_branch> в ветку <branch_name>
git checkout <base_branch>
git merge <branch_name>
```

3. Инструмент для разрешения конфликтов:

```
git mergetool
```

4 Детальная инструкция по выполнению работы

OpenCV – открытая библиотека алгоритмов компьютерного зрения.

Библиотека OpenCV обладает 20-летней историей, и если вам интересно познакомиться с вехами ее развития, то можно прочитать про этапы ее становления на хабре: статьи «OpenCV — 20! Второй проект центра разработки Intel в России» (https://habr.com/ru/company/intel/blog/507382/) и «Как открывали глаза компьютерам» (https://habr.com/ru/post/561052/), которые содержат много воспоминаний разработчиков из истории библиотеки.

Все алгоритмы, представленные в библиотеке OpenCV, сгруппированы в модули:

- **core** – ядро библиотеки, содержащее основные типы данных и математические функции
- **imgproc** – модуль обработки изображений (фильтрация изображений, функции рисования, цветовые пространства)
- **video** – модуль видеоаналитики
- **features2d** – модуль, содержащий реализацию детекторов и дескрипторов ключевых точек изображения
- **objdetect** – модуль детектирования объектов с помощью каскадных классификаторов
- **ml** – модуль классических алгоритмов машинного обучения (кластеризация, регрессия, статистическая классификация)
- **dnn** – модуль для вывода глубоких нейронных сетей

Пожалуй, самые часто используемый модуль библиотеки OpenCV – модуль обработки изображений. В данном туториале мы рассмотрим функции для чтения и записи изображений, изменения размера и конвертации в разные цветовые пространства.

Последовательность действий для выполнения работы.

1. Прочитать изображение
2. Преобразовать цветное изображение в оттенки серого
3. Нормализовать изображение в оттенках серого
4. С помощью детектора Хаара найти изображение кошачьей мордочки на изображении
5. Вывести обнаруженный объект на экран
6. Вырезать обнаруженный объект и сохранить как отдельное изображение

Для реализации первой практики подготовлен шаблон **cat_passport.py**, в котором нужно написать недостающий код.

Файл **cat_passport.py**

```
import argparse
import cv2


def make_cat_passport_image(input_image_path, haar_model_path):

    # Read image

    # Convert image to grayscale

    # Normalize image intensity

    # Resize image if needed

    # Detect cat faces using Haar Cascade

    # Draw bounding box

    # Display result image

    # Crop image

    # Save result image to file

    return


def build_argparser():
    parser = argparse.ArgumentParser(
        description='Speech denoising demo', add_help=False)
    args = parser.add_argument_group('Options')
    args.add_argument('-h', '--help', action='help',
default=argparse.SUPPRESS,
                      help='Show this help message and exit.')
    args.add_argument('-m', '--model', type=str, required=True,
                      help='Required. Path to .XML file with pre-trained
model.')
    args.add_argument('-i', '--input', type=str, required=True,
                      help='Required. Path to input image')
    return parser


def main():

    args = build_argparser().parse_args()
    make_cat_passport_image(args.input, args.model)

    return 0


if __name__ == '__main__':
```

```
    sys.exit(main() or 0)
```

Для запуска вашей программы на исполнение необходимо выполнить следующую команду:

```
python cat_passport.py –i “path_to_image” –m “path_to_haar_model”
```

Для чтения изображений в библиотеке OpenCV используется функция **imread.**

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

Многие алгоритмы работают только с одноканальными изображениями, поэтому необходимо сконвертировать изображение в оттенки серого. Для этого используется функция **cvtColor**.

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

Для работы алгоритмов алгоритмов компьютерного зрения данные часто нормализуются. Для модели детектирования Хаара нужно нормализовать яркость пикселей изображений.

```
gray = cv2.normalizeHist(gray)
```

Размер изображения влияет на скорость его обработки, а порой и на результат работы, поэтому измените размер изображения на 640*480, что довольно близко к данным, на которых обучалась модель. Также, натренированные каскады Хаара очень плохо воспринимают непропорциональные изображения, поэтому старайтесь сохранить соотношение сторон при изменении размера. Изображение в репозитории размера 600*400, его размер изменять не требуется.

```
resized = cv2.resize(gray, (640, 480), interpolation = cv2.INTER_AREA)
```

Для детектирования кошачьей мордочки мы воспользуемся загрузим натренированный детектор Хаара. Детектор Хаара относится к алгоритмам классического машинного обучения и компьютерного зрения.

```
detector = cv2.CascadeClassifier(model)
rects = detector.detectMultiScale(image, scaleFactor=1.1, minNeighbors=5,
minSize=(75, 75))
```

Для вывода на экран изображения, необходимо создать графическое окно и отрисовать в нем изображение. Наша программа однопоточная, изображение будет отрисовывать до тех пор, пока не нажата какая–либо клавиша на клавиатуре, после чего все графические окна будут уничтожены. С помощью функции imshow можно открыть несколько разных окон, если задать им разные имена.

```
cv2.imshow("window_name", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Перед выводом изображения на экран, с помощью функции rect нарисуйте ограничивающий прямоугольник вокруг задетектированных объектов на выводимо изображении.

```
for (i, (x, y, w, h)) in enumerate(rects):
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)
    cv2.putText(image, "Cat #{}".format(i + 1), (x, y - 10),
        cv2.FONT_HERSHEY_SIMPLEX, 0.55, (0, 0, 255), 2)
```

Для того, чтобы обрезать изображение, в C++ используется функция crop, а в Python мы можем использовать концепцию numpy срезов (оригинальный термин slicing, https://coderlessons.com/tutorials/python-technologies/uchitsia-numpy/numpy-indeksirovanie-i-narezka)

```
x, y, w, h = rects[0]
image = image[y:y+h, x:x+w]
```

Сохраните изображение с помощью функции imwrite. Если вы указали относительный путь до файла, то корневой папкой будет является папка, установленная текущей в вашей консоли.

```
cv2.imwrite('out.jpg', image)
```

Для того, чтобы загрузить ваши изменения на github, сначала создайте новый коммит с вашими изменениями и отправьте его в свой GitHub репозиторий, а затем сделайте пул-реквест через сайт.

5 Дополнительные задания

Созданный пример классификации содержит минимально необходимый функционал. В качестве дополнительных заданий предлагается обеспечить поддержку следующих возможностей:

1. С помощью функций OpenCV откройте изображение Литература

6 Литература

1. Шолле Ф. Глубокое обучение на Python. – СПб.: Питер. – 2018. – 400с.
2. Рамальо Л. Python. К вершинам мастерства / Пер. с англ. Слинкин А.А. – М.: ДМК Пресс. – 2016. – 768с.
3. Открытый курс по OpenCV [https://opencv.org/opencv-python-free-course/]
4. Страница скачивания Python 3.8.9 [https://www.python.org/downloads/release/python-389].