можливостей під час 3D-моделювання героїв відеоігор у майбутньому адже фотограмметрія дозволяє зробити їх більш точними, реалістичними та чіткими. Також завдяки використанню попередньо розробленого методичного забезпечення, а саме відеоуроків, процес вивчення теми став досить легким та цікавим і це дозволяє зробити висновок, що процес застосування розробленої методики навчання студентів зі створення та анімування 3D-моделей на основі фотографій засобами цифрових технологій є досить ефективним.

ЛІТЕРАТУРА

1. Все про створення 3d-моделей за фотографіями. URL : https://jak.koshachek.com/articles/vse-pro-stvorennja-3d-modelej-za-fotografijami.html (дата звернення: 15.06.2022).
2. Гаманець А.О, Дейнеко Ж.В. Сплайнове моделювання при розробці тривимірної сцени: Матеріали семінару І Міжнародної наук.-техн. конф. Молодіжної школи, м. Харків, 16-20 травня 2016 р. Харків 2016. С. 79-82. (дата звернення: 21.11.2021).
3. Муха Н.О. Відеоурок – як інноваційна форма навчально - виховного процесу. URL : https://nmc-pto.dp.ua/doc/2016/mkab_47.pdf (дата звернення: 10.11.2021).
4. Тривимірна графіка – 3D-моделювання. URL : https://easy3dprint.com.ua/uk/trivimirna-grafika-3d-modelyuvannya/ (дата звернення: 09.11.2021).
5. Цуканова А.М. 3D моделювання для дітей. URL : https://buki.com.ua/news/try-d-modelyuvannya/ (дата звернення: 03.12.2021).

Євгенія Вітвіцька
Науковий керівник – доц. Тарас Сіткар

ПРОЕКТУВАННЯ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ДЕТЕКТУВАННЯ ОБЛИЧ

Зображення складають велику кількість даних, які генеруються щодня, тому вміння їх обробляти є дуже важливим. Одним із методів обробки зображень є розпізнавання облич. Розпізнавання облич – це область обробки зображень, яка використовує машинне навчання для пошуку облич на зображеннях.

Каскади Хаара (Haar Cascade) – це метод виявлення об’єктів, який використовується для визначення місцезнаходження об’єктів на зображеннях. Алгоритм навчається на великій кількості позитивних і негативних зразків: позитивні зразки – це зображення, які містять об’єкт які нас цікавлять, а негативні – зображення, які містять будь-що, крім об’єкта який нас цікавить. Після навчання класифікатор може знайти об'єкт який нас цікавить на нових зображеннях.

У даній статті ми будемо використовувати попередньо навчену модель Хаара, OpenCV і Python для виявлення та виділення облич із зображення. OpenCV — це бібліотека програмного забезпечення з відкритим кодом, яка використовується для обробки зображень.

Вимоги до середовища розробки:

Для роботи потрібно підготовлене середовище Python 3, включаючи pip і venv.

Налаштування локального середовища

Перш ніж почати писати будь-який код, ми повинні спочатку підготувати своє робоче середовище та встановити деякі залежності.

Створюємо каталог для проекту:

```
mkdir face_scrapper
```

Далі переходимо в нього:

```
CD face_scrapper
```

Потім потрібно створити віртуальне середовище для цього проекту. Віртуальне середовище ізолює різні проекти, розміщені в одній операційній системі, щоб їхні залежності не конфліктували.

Створюємо віртуальне середовище під назвою face_scrapper :

```
python 3 -m venv face_scrapper
```

Активуємо ізольоване середовище:

```
source face_scraper/bin/enable
```

Тепер в командному рядку з'явиться префікс - ім'я нашого віртуального середовища:

```
(face_scrapper) 8hosts-MPB:~ /face_scrapper$
```

Після активації віртуального середовища ми можемо використовувати nano або інший текстовий редактор для створення необхідного файлу require.txt. Цей файл визначає залежності Python .

```
nano requirements.txt
```

Далі потрібно встановити три залежності:

• numpy: це бібліотека Python , яка забезпечує підтримку великих багатовимірних масивів. Вона також містить великий набір математичних функцій для роботи з масивами.

• opencv-utils: розширена бібліотека для OpenCV, яка містить службові функції.

• opencv-python: основний модуль OpenCV, що використовує Python.

Додамо такі залежності до файлу:

```
numpy
opencv-utilits
opencv-python
```

Зберігаємо і закриваємо файл.

Встановлюємо залежності, передавши файл requirements.txt до менеджера пакетів Python pip. Прапорець -r визначає розташування файлу requirements.txt.

```
pip install -r requirements.txt
```

Отже, віртуальне середовище проекту активовано, потрібні бібліотеки встановлені. Тепер ми можемо писати код для виявлення облич на вхідних зображеннях.

Написання та запуск детектора обличчя

Зараз ми напишемо код, який приймає зображення як вхідні дані та повертає дві речі:

• Кількість облич, знайдених у вхідному зображенні.

• Нове зображення з прямокутником навколо кожного виявленого обличчя.

Створимо файл для збереження коду:

```
nano app.py
```

У цьому новому файлі почнемо писати свій код, спочатку імпортувавши необхідні бібліотеки. Тут є два модулі для імпорту: cv2 і sys. Модуль cv2 імпортує бібліотеку OpenCV у програму, тоді як sys імпортує звичайні функції Python, які використовуватиме ваш код (наприклад, argv).

```
import cv2
import sys
```

Далі нам потрібно вказати, що вхідне зображення буде передано як аргумент сценарію під час виконання. Метод читання першого аргументу в Python полягає в присвоєнні змінній значення, яке повертає функція sys.argv[1]:

```
imagePath = sys.argv[1]
```

Загальною практикою обробки зображень є спочатку перетворення вхідного зображення на градації сірого. Це тому, що визначення яскравості, на відміну від кольору, зазвичай дає кращі результати під час виявлення об’єктів. Додаємо наступний код, щоб прийняти вхідне зображення як аргумент і перетворити його на градації сірого:

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

Функція .imread() приймає вхідне зображення, передане як аргумент сценарію, і перетворює його на об’єкт OpenCV. Потім функція OpenCV.cvtColor() перетворює вхідний об’єкт зображення на об’єкт у відтінках сірого.

Тепер, коли ми додали код для завантаження зображення, нам потрібно додати код розпізнавання обличчя до файлу:

```
faceCascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
faces = faceCascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=3, minSize=(30, 30))
print("Found {0} Faces!".format(len(faces)))
```

Цей код створює об’єкт faceCascade, який завантажує файл Haar Cascade за допомогою методу cv2.CascadeClassifier. Це дозволяє Python і коду використовувати каскад Хаара.

Потім код застосовує .detectMultiScale() до об’єкта faceCascade. Це створює список прямокутників для всіх виявлених облич на зображенні. Список прямокутників — це набір масивів пікселів у формі Rect(x, y, w, h).

Ось список інших параметрів, які використовуються в цьому коді:

• gray: вказує на використання попередньо завантаженого об’єкта зображення у відтінках сірого OpenCV.

• scaleFactor: цей параметр визначає швидкість, з якою зображення зменшується під час кожного масштабування. Модель використовує фіксований масштаб під час навчання, тому вхідні зображення можна зменшити для кращого виявлення. Цей процес завершується після досягнення порогу, визначеного maxSize і minSize.

• minNeighbors: цей параметр визначає, скільки сусідів (або співпадінь) повинен мати кожен кандидат-прямокутник, щоб зберегти його. Більше значення може призвести до меншої кількості хибних спрацьовувань, але занадто високе значення може виключити справжні спрацьовування.

• minSize: дозволяє визначити мінімально можливий розмір об’єкта в пікселях. Об'єкти, менші за цей параметр, ігноруються.

Після створення списку прямокутників площі підраховуються за допомогою функції len. Після виконання сценарію в результаті повертається кількість розпізнаних облич.

Тепер скористаємось методом .rectangle(), щоб намалювати прямокутник навколо виявленого обличчя:

```
for (x, y, w, h) in faces:
        cv2.rectangle(image,(x,y),(x+w,y+h),(0,255,0),2)
```

У цьому коді використовується цикл for для перегляду списку розташувань пікселів, повернутого методом faceCascade.detectMultiScale для кожного виявленого об’єкта. Метод прямокутника приймає чотири аргументи:

• зображення малює прямокутники на вихідному зображенні.

• (x, y), (x+w, y+h) — це розташування чотирьох пікселів виявленого об’єкта. Метод прямокутника використовує їх для малювання прямокутника навколо знайденого обличчя на вхідному зображенні.

• (0, 255, 0) встановлює колір фігури. Цей аргумент передається як кортеж BGR. Наприклад, щоб зробити прямокутник синім, ви повинні використати (255, 0, 0). Використовуємо зелений.

• 2 - ширина лінії в пікселях.

Тепер, коли ми додали код для малювання прямокутників, використовуємо .imwrite(), щоб записати нове зображення до локальної файлової системи як face_detected.jpeg. Цей метод повертає true, якщо захоплення було успішним, і false, якщо захоплення нового зображення не вдалося.

```
status = cv2.imwrite('faces_detected.jpg', image)
```

Тепер додамо цей код, щоб надрукувати true або false статус функції .imwrite(). Це повідомить нам, чи вдалося захоплення зображення після запуску сценарію.

```
print("Image faces_detected.jpg written to filesystem: ", status)
```

В результаті ми отримали наступний скрипт:

```
import cv2
import sys
imagePath = sys.argv[1] image = cv2.imread(imagePath)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
faceCascade = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")
faces = faceCascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=3,
minSize=(30, 30))
faces · = · faceCascade.detectMultiScale(gray, · scaleFactor=1.3, · minNeighbors=3,
minSize=(30, ·30))¶
print("[INFO] ·Found ·{0} ·Faces!".format(len(faces)))¶
for ·(x, ·y, ·w, ·h) ·in ·faces:¶
cv2.rectangle(image, (x,y), (x+w,y+h), (0,255,0), 2)¶
status ·= ·cv2.imwrite('faces_detected.jpg', ·image)¶
print("[INFO] ·Image ·faces_detected.jpg ·written ·to ·filesystem: ·", ·status)¶
```

Переконавшись, що все введено правильно, зберігаємо і закриваємо файл. Наш код готовий.

Запуск скрипта

Тепер нам знадобиться тестове зображення, щоб перевірити наш сценарій. Якщо ми знайшли зображення, яке хочете використати для тестування, зберігаємо його в тому ж каталозі, що й сценарій app.py.

```
curl ·-O ·https://path/to/input_image¶
```

Отримавши фотографію, на якій потрібно розпізнавати обличчя, запускаємо скрипт і вказуємо шлях до зображення:

```
python app.py path/to/input_image
```

Ми повинні отримати наступний результат:

```
[INFO] Found 4 Faces!
[INFO] Image faces_detected.jpg written to filesystem: True
```

Виведення на екрані true вказує на те, що оброблене зображення було успішно збережено у файловій системі. Відкрийте зображення на локальному комп'ютері, щоб побачити зміни в новому файлі.

Ви повинні побачити, що ваш сценарій виявив обличчя у вхідному зображенні та намалював прямокутники для їх позначення. Тепер спробуйте використати ці пікселі, щоб виділити обличчя із зображення.

Отримання обличчя та зберігання його на локальному комп'ютері

Раніше ми писали код OpenCV і каскад Хаара для виявлення та малювання прямокутників навколо облич на зображенні. Зараз ми трохи змінимо код, щоб отримувати виявлені обличчя із зображення в окремі файли.

Відкриваємо app.py у текстовому редакторі:

```
nano app.py
```

Після рядка cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2) вставляємо наступні стрічки:

```
roi_color = image[y:y + h, x:x + w]
print("[INFO] Object found. Saving locally.")
cv2.imwrite(str(w)+str(h)+'_faces.jpg',roi_color)
```

Об’єкт roi_color вказує позиції пікселів у списку граней у вихідному вхідному зображенні. Змінні x, y, h і w є розташуванням пікселів для кожного об'єкта, виявленого методом faceCascade.detectMultiScale. Потім код повертає результат, що об’єкт знайдено та зберігається локально.

Після цього код зберігає знайдений об’єкт як нове зображення за допомогою методу cv2.imwrite(). Він додає ширину та висоту об’єкта до імені зображення. Це дасть вам унікальне ім’я файлу (якщо фотографія містить кілька облич).

Оновлений скрипт app.py виглядає так:

```
import cv2
import sys
imagePath = sys.argv[1] image = cv2.imread(imagePath)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
faceCascade = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")
faces = faceCascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=3,
minSize=(30, 30))
print("[INFO] Found {0} Faces.".format(len(faces)))
for (x, y, w, h) in faces:
cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
roi_color = image[y:y + h, x:x + w]
print("[INFO] Object found. Saving locally.")
cv2.imwrite(str(w)+str(h)+'_faces.jpg', roi_color)
status = cv2.imwrite('faces_detected.jpg', image)
print("[INFO] Image faces_detected.jpg written to filesystem: ", status)
```

Оновлений код використовує розташування пікселів для вилучення облич із зображення в новий файл. Зберігаємо та закриваємо файл.

Після оновлення коду ми можемо знову запустити скрипт:

```
python app.py path/to/image
```

Щойно наш сценарій завершить обробку зображення, ми побачимо наступне повідомлення:

```
[INFO] Found 4 Faces.
[INFO] Object found. Saving locally.
[INFO] Object found. Saving locally.
[INFO] Object found. Saving locally.
[INFO] Object found. Saving locally.
[INFO] Image faces_detected.jpg written to file-system: True
```

Залежно від кількості облич у вашому зображенні результат буде більшим або меншим.

Після запуску сценарію, якщо переглянути вміст робочого каталогу, ви побачимо великі зображення всіх облич, знайдених на вхідному зображенні.

Тепер наш сценарій може витягувати виявлені об’єкти з вхідного зображення та зберігати їх локально.

Висновок. У цій статті ми описали сценарії OpenCV і Python для виявлення, підрахунку та вилучення облич із вхідного зображення. Ми можемо в майбутньому розширити цей сценарій, щоб знаходити або вивчати різні об’єкти за допомогою інших попередньо підготовлених каскадів Хаара з бібліотеки OpenCV або навчитися навчати свій власний каскад.

ЛІТЕРАТУРА

1. Open Computer Vision library [Електронний ресурс]/ OpenCV team, 2017. – режим доступу: http://opencv.org/, вільний.
2. Технологія розпізнавання обличчя. [Режим доступу - електронне джерело] - https://securityrussia.com/blog/face-recognition.html
3. EigenFaces. [Режим доступу - електронне джерело] - https://delirium-00.livejournal.com/2764.html
4. OpenCV. [Режим доступу - електронне джерело] - http://masters.donntu.org/2018/fknt/makogon/library/article05.htm