КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Звіт
з самостійної роботи
«Визначення облич людей та очей цих людей на сцені»

Виконав
студент 4 курсу
групи ТК-41
факультету комп'ютерних наук
та кібернетики

Некряч Владислав Вадимович

Київ
2023

OpenCV (англ. Open Source Computer Vision Library, бібліотека комп'ютерного зору з відкритим кодом) — бібліотека функцій та алгоритмів комп'ютерного зору, обробки зображень і чисельних алгоритмів загального призначення з відкритим кодом. Бібліотека надає засоби для обробки і аналізу вмісту зображень, у тому числі розпізнавання об'єктів на фотографіях (наприклад, осіб і фігур людей, тексту тощо), відстежування руху об'єктів, перетворення зображень, застосування методів машинного навчання і виявлення загальних елементів на різних зображеннях.

Бібліотека розроблена Intel і нині підтримується Willow Garage[en] та Itseez. Сирцевий код бібліотеки написаний мовою C++ і поширюється під ліцензією BSD. Біндинги підготовлені для різних мов програмування, таких як Python, Java, Ruby, Matlab, Lua та інших. Може вільно використовуватися в академічних та комерційних цілях.

Офіційно проект OpenCV був запущений у 1999 році за ініціативою Intel Research з ціллю розвивати CPU-ресурсомісткі додатки. Основними вкладниками у проект була Intel's Performance Library Team. На перших етапах розвитку OpenCV основними задачами бібліотеки були:

- Розвивати дослідження у напрямку комп'ютерного зору, забезпечуючи добре оптимізований та відкритий код бібліотеки.
- Поширювати знання у сфері комп'ютерного зору, забезпечуючи загальну інфраструктуру, яку б могли розвивати розробники, таким чином код ставатиме більш легким для сприйняття та обміну.
- Розвивати засновані на роботі з комп'ютерним зором комерційні додатки, створюючи не залежну від платформи, оптимізовану та безкоштовну
  бібліотеку. Для цього використовувалася ліцензія, яка не вимагала від таких комерційних додатків бути відкритими.

Перша альфа-версія OpenCV була оприлюднена на IEEE конференції з комп'ютерного зору й розпізнавання образів у 2000 році, і п'ять бета-версій було випущено у період між 2001 і 2005 роками. Перша версія 1.0 була випущена у 2006 році. У середині 2008 року, OpenCV отримала корпоративну підтримку від Willow Garage і знову перейшла у стадію активної розробки. «Пре-релізна» версія 1.1 була випущена у жовтні 2008 року.

Другий великий випуск OpenCV відбувся у жовтні 2009 року. OpenCV 2 включала у себе серйозні зміни у інтерфейсі C++. Ці зміни спрямовані на більш прості, тип-безпечні моделі, додавання нових функцій, і кращу реалізацію існуючих моделей в

плані швидкодії (особливо на багатоядерних системам). Офіційні релізи надалі відбуваються кожні 6 місяців і розробкою займається незалежна команда з Росії, яка підтримується комерційними корпораціями.

У серпні 2012 року, підтримку OpenCV було передано некомерційній організації, OpenCV.org.

Мотивація:

Алгоритми комп'ютерного зору для виявлення облич є важливими з багатьох причин:

1. Безпека: Алгоритми виявлення облич можуть використовуватися в системах безпеки для ідентифікації осіб та контролю їхніх рухів, сприяючи відстеженню та запобіганню злочинної діяльності.

2. Ідентифікація: Алгоритми виявлення облич можуть використовуватися для ідентифікації осіб у різних контекстах, наприклад, у аеропортах, роздрібних магазинах та інших громадських місцях. Це може допомогти поліпшити досвід користувачів та оптимізувати роботу.

3. Спостереження: Алгоритми виявлення облич можуть використовуватися в системах спостереження для виявлення підозрілих дій та повідомлення охоронних служб в режимі реального часу.

4. Соціальні медіа: Алгоритми виявлення облич використовуються в соціальних медіа-платформах для запропонування тегів для фотографій та відео, що допомагає користувачам знайти та організувати свій контент.

5. Медичне зображення: Алгоритми виявлення облич можуть використовуватися в медичному зображенні для аналізу обличчя та виявлення аномалій або захворювань.

Реалізація:

Для реалізації проекту було використано мову Python. Також використовувалися попередньо навчені моделі res10_300x300_ssd_iter_140000.caffemodel для розпізнавання облич та haarcascade_eye для розпізнавання очей.

ResNet10 є моделлю нейронної мережі згорткового типу, що належить до сімейства моделей ResNet (Residual Network). Архітектуру ResNet було запропоновано Не та ін. у 2015 році, і вона розроблена для вирішення проблеми зникнення градієнтів, яка може виникати під час навчання глибоких нейронних мереж. Зокрема, ResNet10 складається з 10 шарів, включаючи згортковий шар, шари пакетної нормалізації, функції активації та шари пулінгу.

Однією з ключових інновацій ResNet є використання "residual" (або "skip") зв'язків. Ідея за "residual" зв'язками полягає в тому, що вони дозволяють градієнту безпосередньо пройти через мережу без значного затухання, що полегшує навчання глибших мереж. "Skip" зв'язки в ResNet10 дозволяють мережі вчити "residual" функції, які додаються до виходу попереднього шару, а не безпосередньо вчити відображення між входом та виходом. Це означає, що мережа може вчити більш складні відображення, використовуючи менше параметрів, що призводить до кращої ефективності на різних завданнях.

Ще однією важливою особливістю ResNet10 є використання пакетної нормалізації. Пакетна нормалізація допомагає покращити стабільність та швидкість навчання, нормалізуючи вхід до кожного шару до нульового середнього та одиничну дисперсію. Це допомагає запобігти тому, що вхідні дані стануть занадто великими або занадто малими та гарантує більш ефективне використання функції активації. У ResNet10 використовується функція активації ReLU (Rectified Linear Unit), яка є нелінійною функцією, що забезпечує нелінійність у мережі.

ResNet10 є легкою та швидкою моделлю, яка може бути використана для багатьох завдань обробки зображень, включаючи класифікацію, виявлення облич, виявлення об'єктів та семантичну сегментацію. Завдяки своїй високій ефективності та легкості використання, ResNet10 є популярним вибором серед дослідників та практиків у галузі комп'ютерного зору. В цій роботі вона розпізнає обличчя людей.

Друга модель — для розпізнавання очей - **Haar Cascade**. Модель була розроблена Віола і Джонсом в 2001 році. Це метод обробки зображень, який використовується для виявлення об'єктів у зображеннях або відео. Він базується на використанні признаків Хаара, які є простими прямокутними шаблонами, що можуть використовуватися для виявлення конкретних ознак об'єкта.

Метод працює наступним чином: спочатку тренується класифікатор, використовуючи набір позитивних та негативних зображень. Позитивні зображення містять приклади об'єкта, який потрібно виявити, тоді як негативні зображення не містять його. Після цього класифікатор використовує навчену модель для сканування зображення або відеокадра з метою виявлення наявності об'єкта.

Під час процесу сканування використовуються призначені признаки Хаара, щоб визначити, чи містить певна область зображення ймовірність того, що в ній міститься об'єкт. Якщо регіон містить признаки, які відповідають моделі класифікатора, він вважається позитивним, інакше - негативним. Таким шляхом сканування всього зображення або відеокадру, класифікатор може визначити місця розташування шуканих об'єктів.

Haar Cascade широко використовується у застосунках, таких як виявлення облич, виявлення пішоходів та відстеження об'єктів. В цій роботі він використовується для пошуку очей людей.

Завантажуємо і підключаємо необхідні моделі.

```
import cv2
if __name__=="__main__":
    modelFile = "res10_300x300_ssd_iter_140000.caffemodel"
    configFile = "deploy.prototxt"
    eye_cascade = cv2.CascadeClassifier("haarcascade_eye.xml")
    net = cv2.dnn.readNetFromCaffe(configFile, modelFile)
```

Відкриваємо зображення(cv2.imread()), записуємо його деякі параметри у змінні. Конвертуємо його у градації сірого(cv2.cvtColor(sampleImage, cv2.COLOR_BGR2GRAY)).

```
sampleImage = cv2.imread("sample0.jpg")
frameOpencvDnn = cv2.imread("sample0.jpg")
grayscaleSample = cv2.cvtColor(sampleImage, cv2.COLOR_BGR2GRAY)


dimensions = frameOpencvDnn.shape
height = dimensions[0]
width = dimensions[1]
```

Конвертуємо зображення у BLOB-структуру, передаємо у нейронну мережу за допомогою функції forward(), яка видає 4-мірну матрицю, де 3 вимір це знайдені обличчя, а у 4 вимірі знаходиться інформація про прямокутний окіл для кожного обличчя.

```
blob = cv2.dnn.blobFromImage(frameOpencvDnn, 1.0, (300, 300), [104, 117, 123],
False, False)
net.setInput(blob)

detections = net.forward()
bboxes = []
conf_threshold = 0.95
```

Ітеруємо по знайденим обличчям, якщо рівень впевненості у коректності пошуку вищий за певний заданий (0.95), то записуємо прямокутні межі знайденого обличчя. Для кожного обличчя ідентифікуємо очі за допомогою eye_cascade.detectMultiScale() та окреслюємо прямокутні межі очей функцією cv2.rectangle().

```
for i in range(detections.shape[2]):
    confidence = detections[0, 0, i, 2]
    if confidence > conf_threshold:
        x1 = int(detections[0, 0, i, 3] * width)
        y1 = int(detections[0, 0, i, 4] * height)
        x2 = int(detections[0, 0, i, 5] * width)
        y2 = int(detections[0, 0, i, 6] * height)

        w = x2-x1
        h = y2-y1

        roi_grayscale = grayscaleSample[y1:y2, x1:x2]
        roi_colored = sampleImage[y1:y2, x1:x2]

        scale = 1.01 + 0.1 * int((x2-x1)/200)
        eyes = eye_cascade.detectMultiScale(roi_grayscale,scale, 10,
                                            minSize = (int(w/6), int(h/8)),
                                            maxSize = (int(w/2), int(h/2.2)))
        for (eyes_x, eyes_y, eyes_w, eyes_h) in eyes:
            cv2.rectangle(frameOpencvDnn, (x1+eyes_x, y1+ eyes_y), (x1 + eyes_x +
eyes_w, y1 + eyes_y + eyes_h), (0, 255, 0), 2)
```

Окреслюємо межі обличчя функцією cv2.rectangle() та записуємо результат у файл jp3.jpg

```
cv2.rectangle(frameOpencvDnn, (x1, y1), (x2, y2), (255, 0, 0), 2)
cv2.imwrite("jp3.jpg", frameOpencvDnn)
```

РЕЗУЛЬТАТИ

1. **Вхідний файл**

Вихідний файл:

2. **Вхідний файл:**

Вихідний файл: