Національний авіаційний університет

Факультет кібербезпеки, комп’ютерної та програмної інженерії

**ЗВІТ**

**по практичному завданню № 3**

Дисципліна: «Супровід підсистем цільового навантаження БПЛА»

Кафедра: прикладної математики

ОС: бакалавр

Спеціальність: 113 «Прикладна математика»

ОПП: «Прикладне програмне забезпечення»

Виконав: здобувач вищої освіти 4 курсу. 451 групи

Архіпов Олексій Тімурович

Перевірив: Олександр Олександрович Козачук

Київ 2024

**Мета:** опанування технологій глибокого навчання для реалізації та

застосування детектора об'єктів YOLO та трекера в задачах комп'ютерного

зору. Розробка навичок конфігурування YOLO для ефективного виявлення

та відстеження об'єктів у реальному часі з використанням відеопотоків.

Вивчення основ роботи з нейронними мережами, а також методів

підвищення точності та швидкості обробки даних.

**Постановка задачі:**

1. Запустити yolov5 на довільному відео-потоці.

2. Прибрати усі класи, що відображаються крім класу «person».

3. Зберегти результат у відео файл.

4. Запустити довільний трекер з бібліотеки opencv.

5. Для першого кадру обрати область, що буде знаходитись.

6. Записати результат у відео файл.

7. Відеофайли прикріпити у classroom разом зі звітом.

**Теоретична частина**

YOLOv5 - це алгоритм виявлення об'єктів в зображеннях та відео, що базується на нейромережі з глибоким навчанням. Цей алгоритм використовує модель YOLO (You Only Look Once), яка дозволяє виявляти об'єкти в реальному часі з високою швидкістю та точністю.

Нижче наведено ключові аспекти YOLOv5:

1. Архітектура мережі: YOLOv5 використовує звичайну архітектуру нейромережі, яка базується на ефективних блоках, таких як CSPDarknet53, PAN та багатошарові багатомасштабні агрегації. Вона має високу точність виявлення об'єктів.
2. Швидкість та продуктивність: YOLOv5 забезпечує високу швидкість виявлення об'єктів, що дозволяє застосовувати його в реальному часі. Це досягається завдяки оптимізації алгоритму та використанню ефективних методів обробки даних.
3. Навчання на великих обсягах даних: YOLOv5 може бути навчений на великих обсягах даних, що дозволяє покращити його точність та здатність виявлення різних об'єктів у різних умовах.
4. Підтримка різних завдань: YOLOv5 може використовуватися для виявлення об'єктів на зображеннях та відео, а також для різних завдань обробки зображень, таких як класифікація та сегментація.
5. Легкість використання та розширення: YOLOv5 має простий та зрозумілий інтерфейс для використання та налаштування. Крім того, його можна легко розширити або модифікувати для вирішення конкретних завдань виявлення об'єктів

OpenCV (Open Source Computer Vision Library) - це бібліотека з відкритим вихідним кодом для обробки зображень та комп'ютерного зору. Вона надає інструменти для роботи з зображеннями та відео, включаючи зчитування, запис, обробку, аналіз та візуалізацію даних.

Ключові аспекти OpenCV:

1. Робота з різними типами даних: OpenCV підтримує роботу з різними типами даних, такими як зображення, відео та потокові відео з камери. Вона може працювати з різними форматами зображень та відео, такими як JPEG, PNG, AVI, MP4 тощо.
2. Великий набір функцій: OpenCV містить великий набір функцій для обробки зображень та відео, включаючи фільтрацію, морфологічні операції, видалення шуму, виявлення країв, відстеження об'єктів, розпізнавання облич, вимірювання, геометричні перетворення та багато іншого.
3. Крос-платформеність: OpenCV є крос-платформенною бібліотекою, що підтримується на різних операційних системах, таких як Windows, Linux, macOS та мобільні платформи.
4. Широке співробітництво з іншими бібліотеками: OpenCV може легко поєднуватися з іншими бібліотеками для обробки даних та машинного навчання, такими як NumPy, SciPy, TensorFlow, PyTorch тощо.
5. Розширені можливості машинного зору: OpenCV надає розширені інструменти для роботи з машинним зором, включаючи виявлення об'єктів, відстеження руху, розпізнавання облич, розпізнавання рук тощо.

Трекер MIL (Multiple Instance Learning) в OpenCV - це алгоритм відстеження об'єктів, який використовується для відстеження об'єктів на відео. Він базується на методі навчання з використанням кількох екземплярів об'єктів та динамічному оновленні моделі для відстеження.

Характеристики трекера MIL в OpenCV:

1. Багатоекземплярне навчання (Multiple Instance Learning): Трекер MIL використовує підходи машинного навчання, зокрема багатоекземплярне навчання, для відстеження об'єктів. Він навчається на позитивних та негативних прикладах об'єкта, що допомагає йому впевненіше відстежувати ці об'єкти у майбутньому.
2. Адаптивне оновлення моделі: Трекер MIL використовує адаптивне оновлення моделі під час відстеження. Це дозволяє трекеру адаптуватися до змін у вигляді об'єкта, освітлення, тіней та інших факторів, що впливають на зовнішній вигляд об'єкта.
3. Висока швидкодія: Трекер MIL зазвичай володіє високою швидкодією в порівнянні з іншими алгоритмами відстеження. Це робить його придатним для застосування в реальному часі для відстеження об'єктів на відео.
4. Робота з різними типами об'єктів: Трекер MIL може використовуватися для відстеження різних типів об'єктів на відео, включаючи обличчя, автомобілі, людей тощо.
5. Застосування в різних областях: Трекер MIL може бути застосований в різних областях, таких як відеоспостереження, віртуальна реальність, аналіз поведінки людей, відстеження руху та інші області, де потрібне відстеження об'єктів.

**Опис програми з використанням Yolov5**

import time

import sys

model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)

У цьому рядку завантажується модель YOLOv5 з використанням PyTorch Hub. Завантажується версія yolov5s, яка є легковаго варіантом, і вона завантажується попередньо навченою.

video = cv2.VideoCapture("1.mp4")

Тут відбувається відкриття відеофайлу з назвою "1.mp4" за допомогою функції cv2.VideoCapture() з OpenCV.

if not video.isOpened():

    print("Could not open video")

    sys.exit()

Після відкриття відеофайлу перевіряється, чи він був відкритий успішно. Якщо відкриття не вдалося, виводиться повідомлення про помилку, і програма завершується за допомогою sys.exit().

ok, frame = video.read()

if not ok:

    print ('Cannot read video file')

    sys.exit()

Цей код читає перший кадр відео та перевіряє, чи читання відбулося успішно. Якщо читання не вдалося, програма виведе повідомлення про помилку та завершиться.

fps\_start\_time = time.time()

fps\_frame\_count = 0

width = int(video.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

height = int(video.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

fps = int(video.get(cv2.CAP\_PROP\_FPS))

out = cv2.VideoWriter('output1.avi', cv2.VideoWriter\_fourcc(\*'FMP4'), 15, (width, height))

Код налаштовує параметри для запису відео у вихідний файл "output1.avi". Він використовує чотири символи для кодування відео (FOURCC), швидкість кадрів 15 FPS та розмір кадру, який відповідає розміру вихідного відео.

while True:

    ok, frame = video.read()

    if not ok:

        break

У цьому циклі кожен кадр відео читається за допомогою функції video.read(). Якщо читання кадру відбулося успішно (позначається значенням ok), код продовжує виконуватися. Якщо читання не вдалося, цикл завершується за допомогою команди break.

    input\_tensor = frame

    with torch.no\_grad():

        predictions = model([input\_tensor])

Кожен кадр відео передається в якості вхідного тензору моделі YOLOv5. Параметр torch.no\_grad() використовується для того, щоб заборонити використання градієнтів у процесі інференсу, що дозволяє прискорити обчислення.

    boxes = predictions.pandas().xyxy[0]

    for ind, box in boxes.iterrows():

        if box["name"] == "person":

            x1, y1, x2, y2 = box['xmin'], box['ymin'], box['xmax'], box['ymax']

            cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), (0, 255, 0), 5)

Прогнози, отримані від моделі, містять інформацію про області (bounding boxes) та класи об'єктів. У цьому циклі кожна область перевіряється на наявність людини ("person"). Якщо область відповідає людині, то вона відмічається зеленим прямокутником на кадрі.

    fps\_frame\_count += 1

    if (time.time() - fps\_start\_time) > 1:

        fps = fps\_frame\_count / (time.time() - fps\_start\_time)

        fps\_text = f"FPS: {round(fps, 2)}"

        cv2.putText(frame, fps\_text, (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2)

        fps\_frame\_count = 0

        fps\_start\_time = time.time()

В цих рядках коду обчислюється кількість кадрів на секунду (FPS) та відображається на кадрі. Змінна fps\_frame\_count використовується для підрахунку кількості кадрів, а fps\_start\_time - для вимірювання часу.

    out.write(frame)

    cv2.imshow('frame', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):

        break

Кожен оброблений кадр записується у вихідне відео за допомогою out.write(frame). Також кожен кадр відображається на екрані за допомогою cv2.imshow(). Якщо користувач натисне клавішу "q", цикл завершиться.

video.release()

out.release()

cv2.destroyAllWindows()

**Опис програми з використанням трекеру з бібліотеки opencv**

import cv2

import sys

tracker = cv2.TrackerMIL\_create()

Створюємо об'єкт трекера MIL за допомогою функції cv2.TrackerMIL\_create().

video = cv2.VideoCapture('1.mp4')

Відкриваємо відеофайл '1.mp4' для читання за допомогою cv2.VideoCapture.

if not video.isOpened():

    print("Could not open video")

    sys.exit()

ok, frame = video.read()

if not ok:

    print ('Cannot read video file')

    sys.exit()

frame = cv2.resize(frame, (0, 0), fx=0.6, fy=0.6)

Змінюємо розмір першого кадру на 60% від оригінального, щоб зменшити обчислювальні витрати.

bbox = cv2.selectROI(frame, False)

ok = tracker.init(frame, bbox)

Вибираємо область інтересу (ROI) на першому кадрі за допомогою функції cv2.selectROI, після чого ініціалізуємо трекер MIL для цієї області.

width = int(video.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

height = int(video.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

fourcc = cv2.VideoWriter\_fourcc(\*'FMP4')

scaled\_width = int(width \* 0.6)

scaled\_height = int(height \* 0.6)

out = cv2.VideoWriter('output2.avi', fourcc, 20.0, (scaled\_width, scaled\_height))

Створюємо об'єкт VideoWriter для запису відео з розмірами 60% від оригінального кадру та з кодеком FMP4.

while True:

    ok, frame = video.read()

    if not ok:

        break

    frame = cv2.resize(frame, (0, 0), fx=0.6, fy=0.6)

    timer = cv2.getTickCount()

    ok, bbox = tracker.update(frame)

    fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer)

У циклі читаємо кожен кадр відеофайлу, змінюємо його розмір на 60% від оригінального та оновлюємо трекер MIL на цьому кадрі. Обчислюємо FPS за допомогою засічки часу.

    if ok:

        p1 = (int(bbox[0]), int(bbox[1]))

        p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))

        cv2.rectangle(frame, p1, p2, (255,0,0), 2)

    else :

        cv2.putText(frame, "Tracking failure detected", (100,80), cv2.FONT\_HERSHEY\_SIMPLEX, 0.75,(0,0,255),2)

Якщо трекінг пройшов успішно, малюємо прямокутник навколо об'єкта на кадрі. Якщо трекінг не вдається, виводимо повідомлення про невдачу трекінгу.

    cv2.putText(frame,"Tracker: MIL", (100,20), cv2.FONT\_HERSHEY\_SIMPLEX, 0.75, (50,170,50),2)

    cv2.putText(frame, "FPS : " + str(int(fps)), (100,50), cv2.FONT\_HERSHEY\_SIMPLEX, 0.75, (50,170,50), 2)

Додаємо текст до кадру з інформацією про використаний трекер та FPS.

    cv2.imshow("Tracking", frame)

    out.write(frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):

        break

Відображаємо оброблений кадр з нанесеними об'єктами та текстом, записуємо кадр у вихідне відео і перевіряємо, чи натиснута клавіша "q" для виходу з циклу.

video.release()

out.release()

cv2.destroyAllWindows()

Результат роботи програми для знаходження людей за допомогою Yolov5 записаний у output1.avi.

Результат роботи трекера записаний у output2.avi.

**Висновки**

У ході виконання лабораторної роботи виконав такі завдання:

1. Запустив yolov5 на довільному відео-потоці.

2. Прибрав усі класи, що відображаються крім класу «person».

3. Зберіг результат у відео файл.

4. Запустив довільний трекер з бібліотеки opencv.

5. Для першого кадру обрав область, що буде знаходитись.

6. Записав результат у відео файл.