**Шевченко Максим, ШІ-2**

Звіт з предмету

**Методи обробки мультимедійної інформації**

Загальний код доступний за посиланням:

**Завдання 1. Виділення об’єктів на відео.**

Посилання на код:

Для виділення об’єктів на відео була використана попередньо натренована нейронна мережа YOLOv5.

YOLO (You Only Look Once - Ви дивитеся тільки один раз) є найпопулярнішою архітектурою, яка використовується для виявлення об’єктів у реальному часі й перша модель, яка реалізована на мобільному пристрої. Алгоритм «дивиться лише раз» застосовується до зображення в тому сенсі, що для прогнозування потрібно тільки один прохід прямого розповсюдження через нейронну мережу.

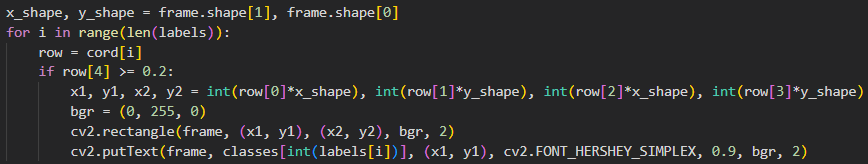
Алгоритм починається з того, що мережа ділить зображення на своєрідну сітку. Залежно від роздільної здатності, сітка має певну кількість клітинок. Потім вона прогнозує відповідну кількість граничних рамок (bounding boxes) для кожної ділянки. Та прогнозує ймовірності того, що в цих рамках є якісь бажаний об'єкт. Кожен граничний блок містить 5 елементів: ширина w, висота h, зміщення до відповідної комірки х і у, а також показник достовірності блоку. Показник достовірності говорить, наскільки мережа впевнена, що передбачуване поле, або рамка насправді містить якийсь об'єкт.

Далі починається етап класифікації і мережа для кожного граничного вікна передбачає клас. Показник достовірності та значення прогнозування класу об’єднуються в один загальний бал, який говорить про ймовірність того, що ця рамка містить конкретний тип об’єкта. YOLO може передбачити кілька граничних рамок на одну клітинку сітки. За допомогою функції помилки ЗНМ знаходить єдине справжнє значення і відкидає рамки, які мають менше значення показника.

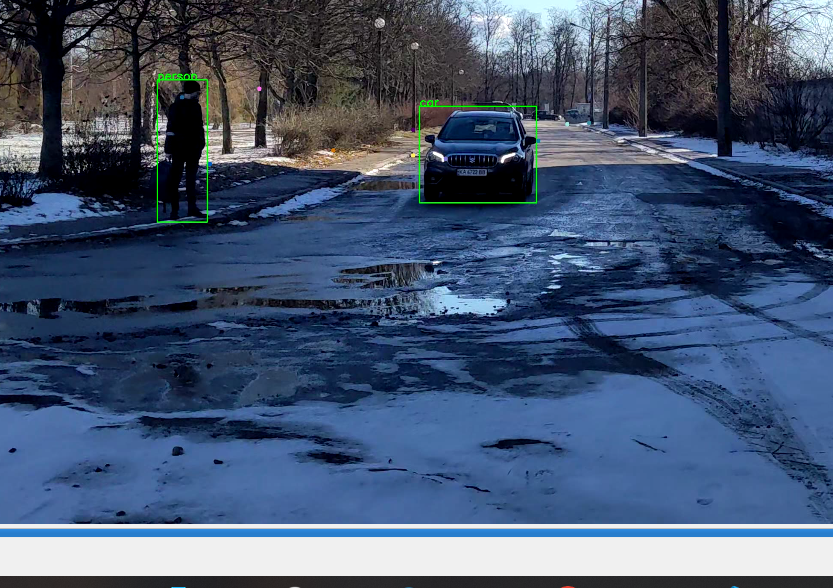
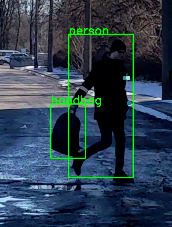
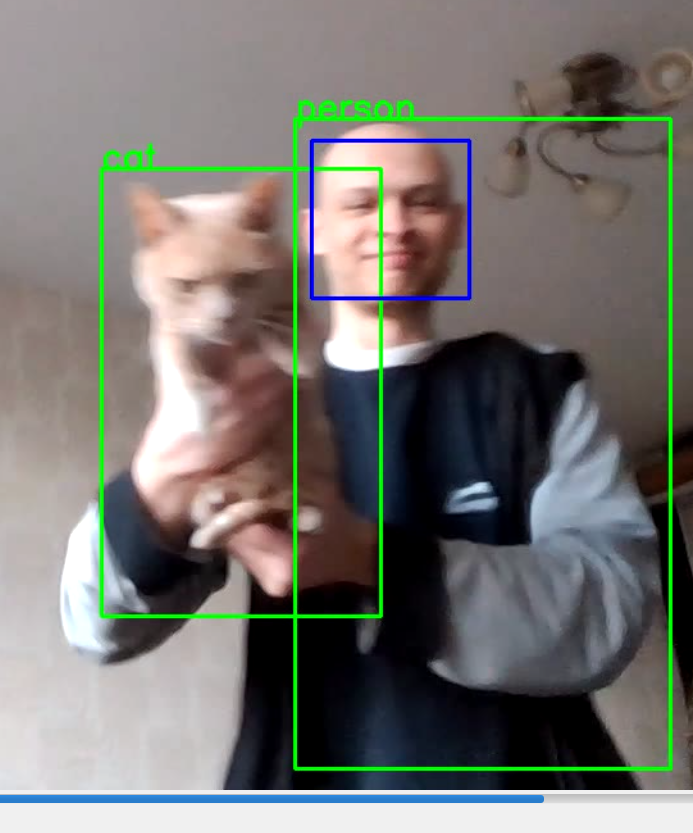
YOLO використовує суму квадратів помилок між прогнозами й істинною величиною для розрахунку втрат. Функція втрат складається з:

* класифікації втрат;
* локалізації втрат (помилки між передбаченою і істинною граничною рамкою);
* втрата довіри (предметність рамки).

Алгоритм застосування моделі YOLO доволі простий:

1. Завантажити мережу з натренованими вагами 
2. Передати фрейм відео в мережу 
3. Візуалізувати знайдені об’єкти 

Результат роботи (виділені прямокутником об’єкти та їхній клас):

**Завдання 2. Виявлення обличчя.**

Посилання на код:

Для виконання цього завдання було використано каскадний класифікатор, який використовує алгоритм Віоли-Джона, алгоритм AdaBoost та Гаароподібні ознаки (Haar-Like Features).

Алгоритм Віоли-Джонса названий на честь двох дослідників комп’ютерного зору, які запропонували цей метод у 2001 році, Пола Віоли та Майкла Джонса у своїй статті «Швидке виявлення об’єктів за допомогою розширеного каскаду простих функцій». Незважаючи на те, що Viola-Jones є застарілою структурою, вона досить потужна, і її застосування виявилося надзвичайно помітним у розпізнаванні облич у реальному часі. Цей алгоритм дуже повільний у навчанні, але може виявляти обличчя в реальному часі з вражаючою швидкістю.

Маючи зображення (цей алгоритм працює із зображенням у градаціях сірого), алгоритм розглядає багато менших субрегіонів і намагається знайти обличчя, шукаючи певні особливості в кожному субрегіоні. Потрібно перевірити багато різних положень і масштабів, оскільки зображення може містити багато граней різного розміру. Віола та Джонс використовували ознаки, подібні до Хаара, для виявлення облич у цьому алгоритмі.

Алгоритм Віоли-Джонс складається з чотирьох основних кроків, які ми обговоримо далі:

1. Вибір Гаароподібних ознак
2. Створення інтегрального зображення
3. Запуск навчання AdaBoost
4. Створення каскадів класифікаторів

Гаароподібні ознаки — це ознаки цифрового зображення, які використовуються для розпізнавання об’єктів. Усі людські обличчя мають деякі універсальні властивості людського обличчя, наприклад область очей темніша за сусідні пікселі, а область носа яскравіша за область очей.

Простий спосіб дізнатися, яка область світліша чи темніша, полягає в тому, щоб підсумувати значення пікселів обох областей і порівняти їх. Сума значень пікселів у темнішій області буде меншою, ніж сума пікселів у світлішій області. Якщо одна сторона світліша за іншу, це може бути край брови або іноді середня частина може бути блискучою за навколишні прямокутники, що можна інтерпретувати як ніс. Це можна зробити за допомогою рис, подібних до Хаара, і за допомогою з них ми можемо інтерпретувати різні частини обличчя.

У своєму дослідженні Віола та Джонс виявили 3 типи ознак, подібних до Хаара:

* Особливості краю
* Лінія-особливості
* Чотиристоронні особливості

Функції країв і ліній корисні для виявлення країв і ліній відповідно. Чотиристоронні елементи використовуються для знаходження діагональних елементів.

Значення функції обчислюється як одне число: сума значень пікселів у чорній області мінус сума значень пікселів у білій області. Значення дорівнює нулю для рівної поверхні, на якій усі пікселі мають однакове значення, і, отже, не надають корисної інформації.

Оскільки наші обличчя мають складну форму з темнішими та яскравішими плямами, функція, подібна до Хаара, дає вам велике число, коли області в чорному та білому прямокутниках сильно відрізняються. Використовуючи це значення, ми отримуємо частину дійсної інформації із зображення.

Щоб бути корисною, функція, подібна до Хаара, повинна давати вам велике число, тобто області в чорному та білому прямокутниках дуже відрізняються. Відомі функції, які дуже добре розпізнають людські обличчя:

Наприклад, коли ми застосовуємо цю особливість хаара до перенісся, ми отримуємо хорошу реакцію. Так само ми об’єднуємо багато з цих функцій, щоб зрозуміти, чи містить область зображення обличчя людини.

Інтегральне зображення відіграє свою роль, дозволяючи нам швидко виконувати ці інтенсивні обчислення, щоб ми могли зрозуміти, чи відповідає критеріям функція кількох функцій.

Інтегральне зображення (також відоме як таблиця сумарної області) — це назва як структури даних, так і алгоритму, який використовується для отримання цієї структури даних. Він використовується як швидкий і ефективний спосіб обчислення суми значень пікселів у зображенні або прямокутній частині зображення.

Далі ми використовуємо алгоритм машинного навчання, відомий як AdaBoost.

Кількість функцій, присутніх у вікні детектора 24×24, становить майже 160 000, але лише деякі з них важливі для ідентифікації обличчя. Тому ми використовуємо алгоритм AdaBoost, щоб визначити найкращі функції серед 160 000 функцій. Щоб визначити тип і розмір функції, яка входить до остаточного класифікатора, AdaBoost перевіряє продуктивність усіх класифікаторів, які ви надаєте йому.

Щоб обчислити продуктивність класифікатора, ви оцінюєте його на всіх підобластях усіх зображень, які використовуються для навчання. Деякі субрегіони дадуть сильну відповідь у класифікаторі. Вони будуть класифіковані як позитивні, тобто класифікатор вважає, що вони містять людське обличчя. На думку класифікаторів, субрегіони, які не дають чіткої відповіді, не містять людського обличчя. Вони будуть класифіковані як негативні.

Класифікаторам, які показали хороші результати, надається більша важливість або вага. Кінцевим результатом є сильний класифікатор, також званий посиленим класифікатором, який містить найефективніші слабкі класифікатори.

Отже, коли ми навчаємо AdaBoost визначати важливі функції, ми передаємо йому інформацію у формі навчальних даних, а потім навчаємо його навчатися на основі інформації для прогнозування. Таким чином, зрештою, алгоритм встановлює мінімальний поріг, щоб визначити, чи можна щось класифікувати як корисну функцію чи ні.

Можливо, AdaBoost нарешті вибере найкращі функції, скажімо, 2500, але обчислення цих функцій для кожного регіону все одно займає багато часу. У нас є вікно розміром 24 × 24, яке ми ковзаємо по вхідному зображенню, і нам потрібно знайти, чи є в одній із цих областей обличчя. Завдання каскаду полягає в тому, щоб швидко відкидати не-обличчя та уникати втрати дорогоцінного часу та обчислень. Таким чином, досягається швидкість, необхідна для виявлення обличчя в реальному часі.

Ми встановили каскадну систему, у якій ми розділили процес ідентифікації обличчя на кілька етапів. На першому етапі у нас є класифікатор, який складається з наших найкращих ознак, іншими словами, на першому етапі субрегіон проходить через найкращі ознаки, такі як ознака, яка ідентифікує перенісся, або та, яка ідентифікує очі. На наступних етапах у нас є всі інші функції.

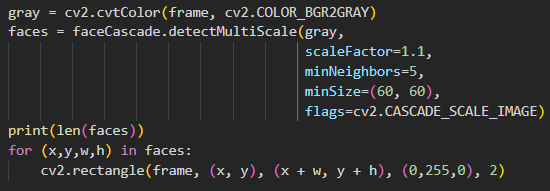
Коли субрегіон зображення потрапляє в каскад, він оцінюється першим етапом. Якщо ця стадія оцінює субрегіон як позитивний, тобто вважає себе обличчям, результатом стадії є можливо.

Коли субрегіон отримує можливо, він переходить до наступного етапу каскаду, і процес продовжується, доки ми не досягнемо останнього етапу.

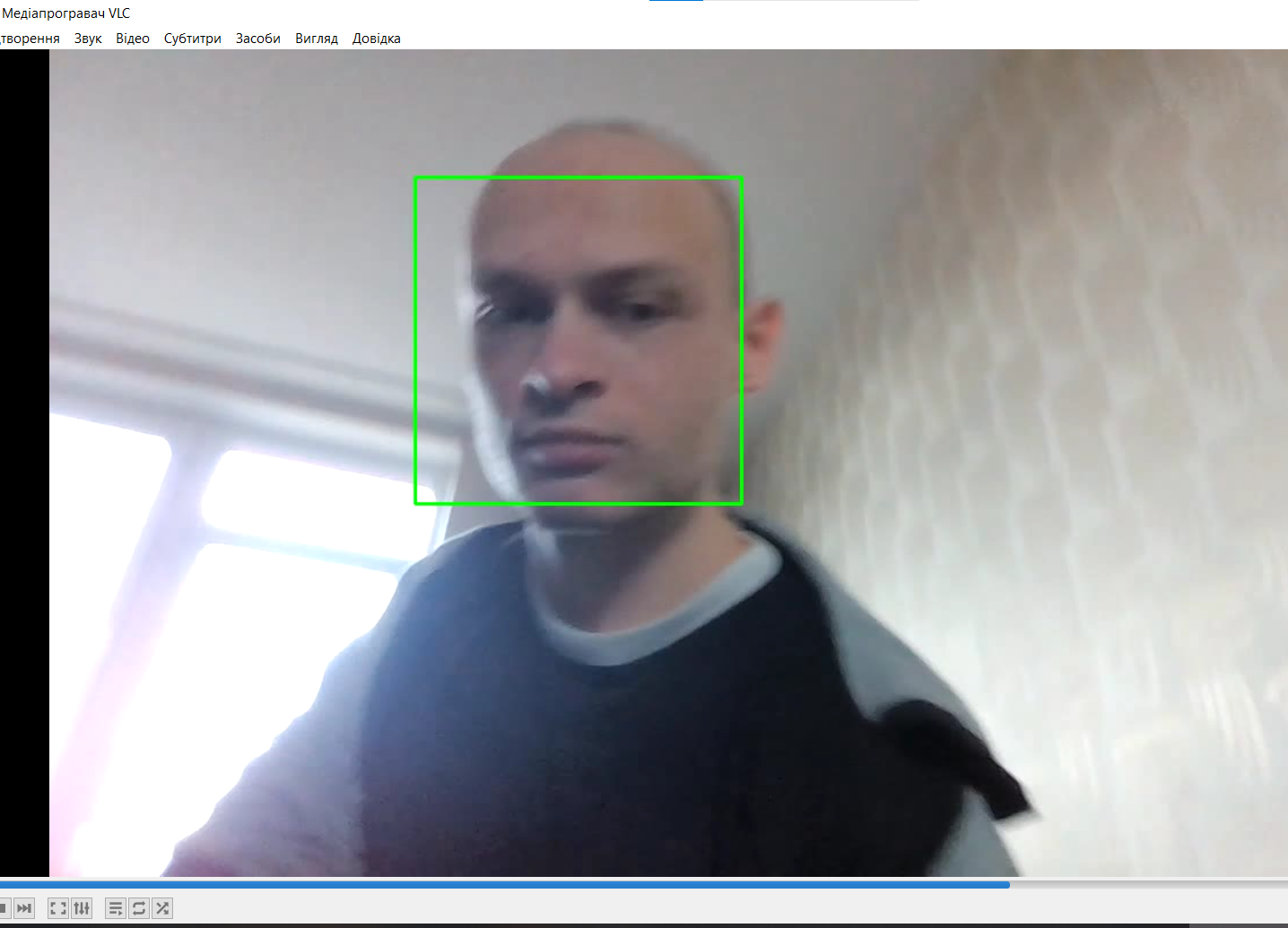
Якщо всі класифікатори схвалюють зображення, воно остаточно класифікується як людське обличчя та представляється користувачеві як виявлення.

Тепер як це допомагає нам збільшити швидкість? Загалом, якщо перший етап дає негативну оцінку, то зображення негайно відкидається як таке, що не містить людського обличчя. Якщо він пройшов перший етап, але не пройшов другий етап, він також викидається. По суті, зображення може бути відкинуте на будь-якому етапі класифікатора.

В роботі був використаний вбудований каскадний класифікатор у бібліотеку opencv-python та використано ваги Гаароподібних ознак для виявлення обличчя.

У цей класифікатор ми передаємо вхідне відео по фреймам та візуалізуємо знайдені обличчя. 

Результат роботи (обличчя, виділене зеленим квадратом на першому фото та синім на другому):

**Завдання 3. Пошук об’єктів у відео потоці.**

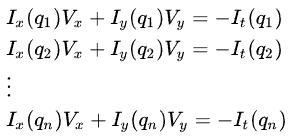
Посилання на код:

У цьому завданні було реалізовано відстеження руху об’єктів за допомогою алгоритму Лукаса-Канаде.

У комп'ютернім зорі, метод Лукаса-Канаде (англ. Lucas-Kanade method, також алгоритм Лукаса-Канаде) — це широко вживаний диференціальний метод оцінювання оптичного потоку, розроблений Брюсом Д. Лукасом та Такео Канаде. Він спирається на припущення, що в локальному околі розгляданого пікселя цей потік є суттєво сталим, і розв'язує базові рівняння оптичного потоку для всіх пікселів у цьому околі методом найменших квадратів.

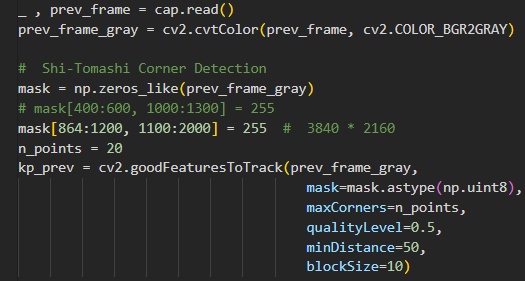
Поєднуючи інформацію з кількох сусідніх пікселів, метод Лукаса-Канаде часто здатен розв'язувати притаманну невизначеність рівняння оптичного потоку. Він також менш чутливий до шуму в зображенні, ніж поточкові методи. З іншого боку, оскільки це суто локальний метод, він не здатен надавати інформацію про потік всередині однорідних областей зображення.

Метод Лукаса-Канаде спирається на припущення, що зміщення вмісту зображення між двома сусідніми моментами (кадрами) є малим і приблизно сталим в межах околу точки p, яку розглядають. Таким чином, можна вважати, що рівняння оптичного потоку виконується для всіх пікселів у межах вікна з центром в p. А саме, вектор локального потоку (швидкості) зображення (Vx, Vy) мусить задовольняти



де  це пікселі всередині вікна, а  це частинні похідні зображення I за положенням за x,y та часом t, оцінювані в точці qi у поточний момент часу. Ця система має більше рівнянь ніж невідомих, і відтак є надвизначеною.

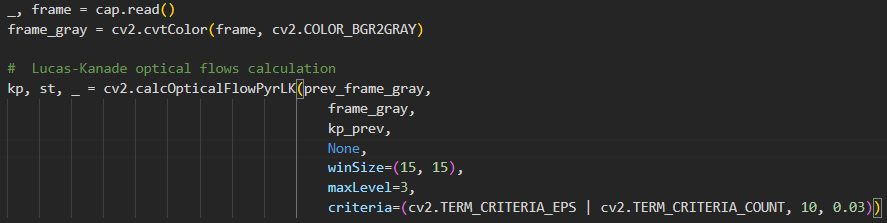
Для застосування цього алгоритму потрібно виконати початкову ініціалізацію ключових точок об’єктів kp\_prev. Для цього було використано алгоритм [Ші-Томаші (Shi-Tomasi Corner Detector)](https://medium.com/pixel-wise/detect-those-corners-aba0f034078b), вбудований у бібліотеку opencv-python (метод cv2.goodFeaturesToTrack):



Тут

* prev\_frame\_gray - вхідне 8-розрядне або 32-розрядне зображення з плаваючою комою, одноканальне зображення
* mask - область інтересу. Якщо зображення не порожнє, вказує область, у якій виявлено кути
* maxCorners - Максимальна кількість кутів для повернення. Якщо кутів більше, ніж знайдено, повертається найсильніший із них. Якщо <= 0 означає, що обмеження на максимум не встановлено, і повертаються всі виявлені кути
* qualityLevel - параметр, що характеризує мінімально допустиму якість кутів зображення
* minDistance – мінімально можлива евклідова відстань між повернутими кутами
* blockSize - розмір середнього блоку для обчислення похідної коваріаційної матриці для кожного піксельного оточення

Далі в метод cv2.calcOpticalFlowPyrLK передається наступний фрейм відео разом із знайденими ключовими точками та попереднім фреймом.



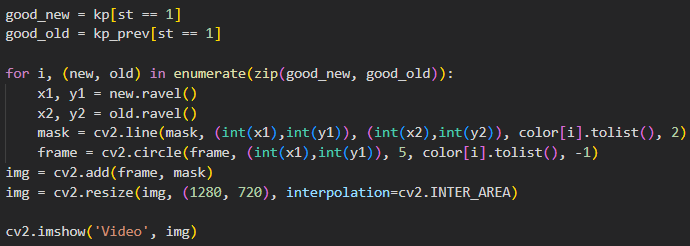
Параметри:

* prev\_frame\_gray – перше 8-бітове вхідне зображення
* frame\_gray – друге вхідне зображення
* kp\_prev – вектор 2D точок, для яких потрібно знайти потік
* nextPts – вихідний вектор 2D точок, тут None
* winSize – розмір вікна пошуку на кожному рівні піраміди
* maxLevel – номер максимального рівня піраміди від 0; якщо встановлено значення 0, піраміди не використовуються (один рівень), якщо встановлено значення 1, використовуються два рівні тощо
* criteria – параметр, що визначає критерії завершення алгоритму ітераційного пошуку

Повертає:

* kp – вихідний вектор 2D точок (з координатами з плаваючою комою одинарної точності), що містить розраховані нові положення вхідних об’єктів на другому зображенні
* st – вихідний вектор статусу (беззнакових символів); кожен елемент вектора встановлюється в 1, якщо потік для відповідних функцій знайдено, в іншому випадку він встановлюється в 0
* err – вихідний вектор помилок; кожен елемент вектора встановлюється на помилку для відповідної функції, тип міри помилки можна встановити в параметрі flags; якщо потік не знайдено, то помилка не визначена

Після чого відбувається візуалізація зміщення:



Результат роботи (кольорова точка, відповідна до об’єкту, та її траєкторія):