Міністерство освіти і науки України
Національний Університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра математики

Кваліфікаційна робота

освітній ступінь - бакалавр

на тему: **«Знаходження і відстеження рухомих об’єктів у відеопотоці»**

Виконав: студент 4-го року навчання
освітньої програми «Прикладна математика»,
спеціальності 113 Прикладна математика
Кольчик Микита Олегович

Керівник:
к.н., доцент *Бучко О.А.*
Рецензент:
к. ф.-м. наук, доцент *Афонін А.О.*

Кваліфікаційна робота захищена
з оцінкою _____

Секретар ЕК _____
(підпис)
«____» _____ 2022 р.

Київ-2022

Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

ЗАТВЕРДЖУЮ
Зав.кафедри математики,
проф., д.ф-м.н., Б. В. Олійник
_____
(підпис)
„____”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на кваліфікаційну роботу

студенту 4-го курсу, факультету інформатики__
Кольчику Микиті Олеговичу_____
Дослідити і порівняти різні алгоритму знаходження і відстеження об’єктів у відеопотоці

Зміст ТЧ до кваліфікаційної роботи:

Зміст
Анотація
Вступ
1 Огляд задачі трекінгу і способів її вирішення
2 Аналіз алгоритму Лукаса-Канаде
3 Аналіз алгоритму MOSSE
4 Аналіз алгоритму DeepSort
5 Порівняння алгоритмів трекінгу у різних умовах
5 Результати порівнянь
Висновки
Список літератури
Додатки

Дата видачі „___” _____ 2021 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Графік підготовки кваліфікаційної роботи до захисту

Графік узгоджено «_____» _____ 2022 р.

№ з/п	Перелік робіт	Термін	Підпис	Дата	Примітка
1.	Отримання теми кваліфікаційної роботи	16.10.2021			
2.	Ознайомлення з існуючою інформацією за темою кваліфікаційної роботи	20.10.2021			
3.	Розробка плану та структури роботи	09.11.2021			
4.	Робота з науковою літературою	11.11.2021			
5.	Аналіз загальновідомих методів трекінгу, їх реалізація	05.01.2022			
6.	Реалізація методу DeepSort	05.02.2022			
7.	Реалізація методу Лукаса-Канаде	01.04.2022			
8.	Реалізація методу Лукаса-MOSSE	10.04.2022			
9.	Аналіз практичної частини, її корегування	17.04.2022			
10.	Початок написання текстової частини	24.04.2022			
11.	Подання проміжної версії текстової частини	10.05.2022			
12.	Остаточне завершення написання текстової частини роботи	21.06.2022			
13.	Створення презентації	22.06.2022			
14.	Захист кваліфікаційної роботи	04.07.2022			

Зміст

Анотація

У цій роботі будуть розглянуті різні за своєю природою алгоритми трекінгу. У першому розділі розглянуто і проаналізовано такі алгоритми трекінгу як алгоритм Лукаса-Канаде і MOSSE(Minimum Output Sum of Squared Error). У другому розділі розглянули алгоритми детекції об’єктів на основі нейронних мереж і алгоритм трекінгу DeepSort. У третьому розділі порівняли всі алгоритми роботи на різних типах відео і зробили висновки щодо їх роботи.

Вступ

Математичні алгоритми машинного навчання кожного року відкривають нові типи алгоритмів або їх вдосконалення. Особлива увага приділяється методам комп'ютерного зору. новітні способи застосування алгоритмів роботи з зображеннями або відео. Найбільш популярні тут- алгоритми комп'ютерного зору. Вони мають надзвичайно широкий спектр застосувань. У сфері бізнесу це – 3D моделювання, теплова карта найбільш відвідуваних місць в магазині. У державній сфері – розпізнавання номерів авто при перевищенні швидкості, пошук людей зі зброєю для забезпечення громадського порядку. Ще однією перспективною задачею є самохідні автомобілі (self-driving cars). Особливістю в роботі таких автомобілів є те, що їх радари і камери повинні розпізнавати навколишні об'єкти для того, щоб приймати ті або інші рішення на основі отриманої інформації про них.

Для нас цікавою буде проблема знаходження та відстеження об'єктів у відео. В основному такими відео будуть відео з камер спостереження. Оскільки проблема розпізнавання об'єктів не нова і існує ціла низка алгоритмів для детекції об'єктів, у цій роботі ми розглянемо абсолютно різні підходи до трекінгу, пов'язані як із рухом зображення, так і з унікальними характеристиками відстежуваного об'єкта. Варто зазначити, що не всі з них працюють швидко, тому в роботі будуть state-of-the-art рішення, хоча це не означає, що це будуть ідеальні алгоритми. Як і в усіх алгоритмах машинного навчання, ми повинні знаходити золоту середину між якістю і швидкістю, тому ми опиратимемося на швидкість.

Отже, метою роботи буде опис і порівняння алгоритмів трекінгу там, де це можливо. Також спробуємо побудувати траєкторії руху відстежуваних об'єктів. Варто зазначити, що траєкторії будуть об'єктивними тільки при нерухомій камері, бо з рухом камери зсується і траєкторія.

Актуальність

Інтерес до розпізнавання образів з кожним роком зростає. Це можна побачити на графіку згадувань з Google Ngram Viewer:

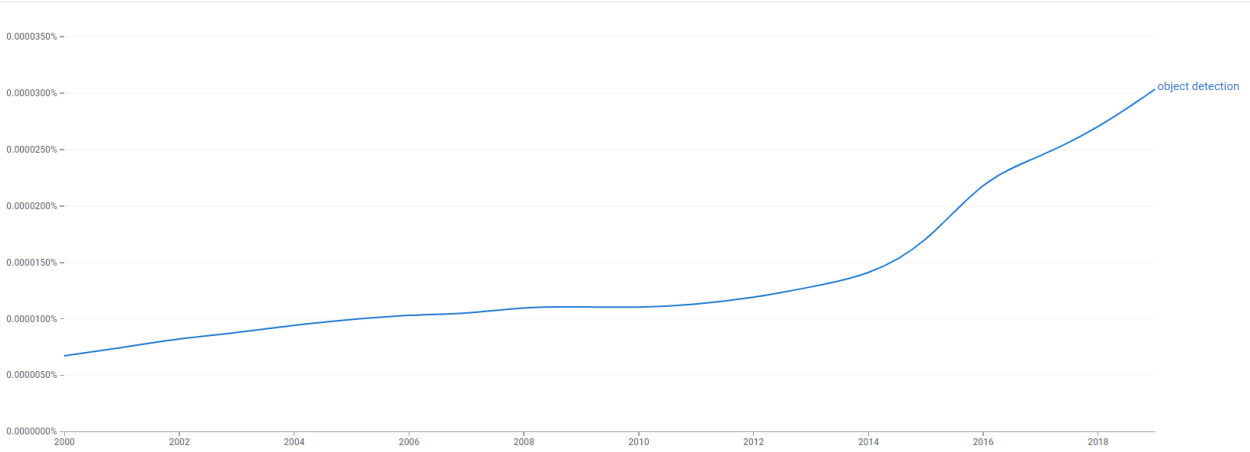


Рис. 1 – відсоток згадувань фрази "Object detection" [16]

Також графік з сайту researchgate показує щорічний ріст кількості публікацій щодо тами object detection:

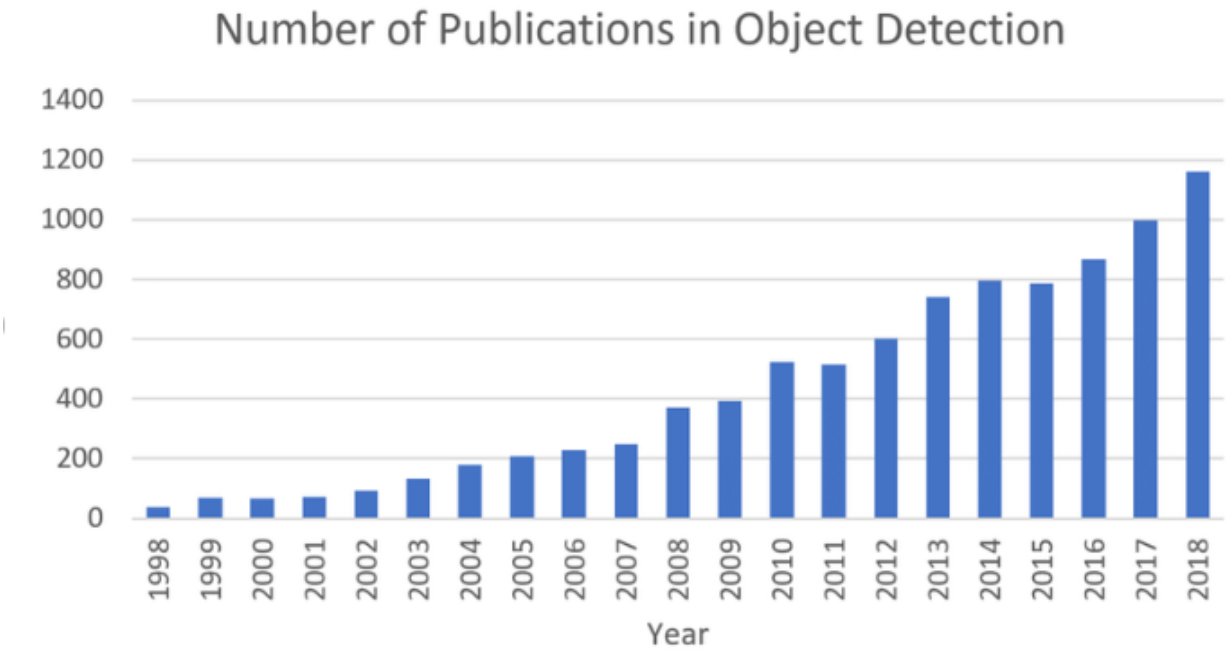


Рис. 2 - Кількість публікацій на researchgate [17]

Отже, даний розділ математики має неабиякий потенціал, що робить цю тему надзвичайно актуальною.

1. Трекінг об’єктів без попереднього навчання

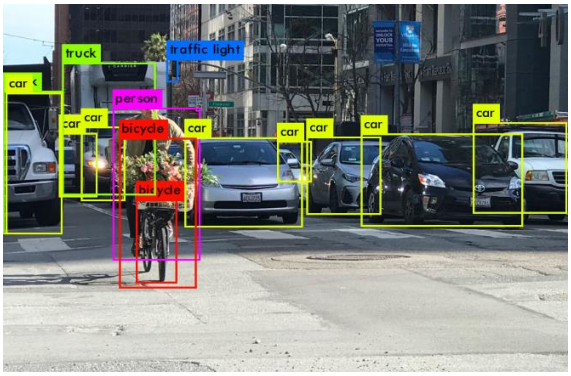
1.1 Object detection і object tracking

Насправді, ці дві задачі дещо схожі, бо трекінг об’єктів напряму залежить від того, чи знаємо який об’єкт ми відстежуємо і чи знаємо де він. Грубо кажучи, трекінг є доповненням до детекції разом з траєкторією, яку і визначають алгоритми трекінгу.

Завдання object detection – ідентифікувати об’єкт і вказати його локацію. Для візуального сприйняття, навколо нього обводять обмежувальну рамку у вигляді прямокутника, довжина і ширина якого будуть довжиною і шириною знайденого об’єкта. Такий прямокутник називається bounding box, в надалі він буде позначатися як bbox. Оскільки ми працюватимемо з відеопотоком, на кожному фреймі bbox-и визначатимуться окремо. Зазвичай до bbox-а додають впевненість bbox-а, тобто імовірність того, що bbox є правильним.

Але на фреймі t, bbox навколо об’єкта A є незалежним від bbox-а на фреймі $t + 1$. Тому задачею object tracking буде відслідкувати всі bbox-и для кожного об’єкту, об’єднати їх і, відповідно, побудувати траєкторії їх руху. [1]

Типові проблеми для обох напрямів- перетин спостережуваного об’єкта з іншим об’єктом, поворот об’єкта, зміна масштабу об’єкта.



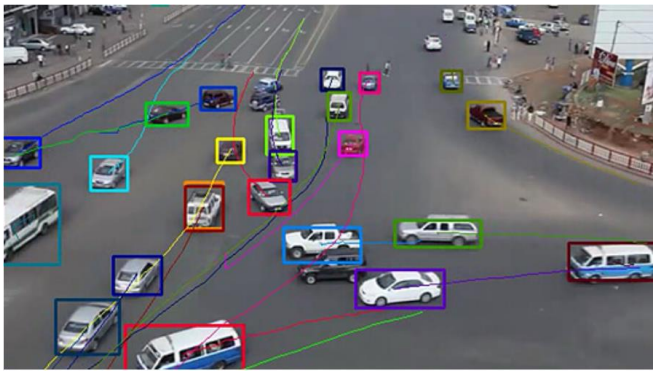


Рис. 1.1, 1.2 - Приклад роботи алгоритмів Object detection і Object tracking [18],[19]

1.2 Особливості різних типів алгоритмів трекінгу

Алгоритми для знаходження і відстеження об’єктів поділяються на такі, що працюють без попереднього навчання, і такі, що потребують попередні знання (Deep Learning-based).

Серед алгоритмів, які працюють без навчання, досить непогану якість показують такі, що засновані на аналізі змін між кадрами, тобто на аналізі оптичного потоку. Серед таких алгоритмів є алгоритм Лукаса-Канаде (Lucas-Kanade), алгоритм Хорна- Шанка (Horn-Schunck), алгоритм Фарнебека (Farnebeck) і алгоритми на кореляційних фільтрах.

Їхня особливість полягає в тому, що вони визначають рух всіх об’єктів у всьому відеопотоці, що може бути шкідливим для наших результатів. Тобто, алгоритм визначить рух маленьких і непотрібних об’єктів, які можна віднести до шуму.

Для того, щоб позбавитися такого шуму, часто в таких алгоритмах області, які нам необхідно відстежити, задають вручну. Такі області називають region of interest (ROI). На початку відеопотоку, ми виділяємо точку з околом. В такій ROI знаходиться об’єкт, який нам цікавий для відстеження, і тому ми не матимемо зайвих об’єктів для відстеження.

Алгоритми з глибоким навчанням повинні бути натреновані на датасетах для того, щоб вилучити основні риси з класів об’єктів, які мають бути знайдені і відстежені. Найпопулярніші класи об’єктів- Людина, Автомобіль, Тварина, інші. Для відмінного знаходження цих класів на зображеннях, фахівці навчають ці алгоритми на тисячах зображень, тому алгоритм здатен виявляти один об’єкт, якщо він є повернутий, зменшений, з шумом. Найвідоміші алгоритми тут – MDNet і DeepSort.

Є також відмінності у застосуванні- алгоритми без навчання працюють краще, коли ми самі відмічаємо точку з бажаним околом або область, яку ми

хочемо відстежити. В той час як алгоритми з навчанням самі знаходять необхідні області для трекінгу.

1.3 Огляд бібліотеки OpenCV

OpenCV- бібліотека на C++, яка реалізовує багато методів з комп'ютерного зору. Також доступна на мовах Python і Java. Вона складається з чотирьох основних модулів: [2]

- CXCORE – ядро, яке містить базові структури і алгоритми
- CV – обробка зображень і алгоритми комп'ютерного зору
- MLL – компонента з алгоритмами машинного навчання
- HighGUI – функції завантаження і збереження відео/зображень, робота з графічним інтерфейсом
- CvAux – містить експериментальні алгоритми

1.4 Реалізовані алгоритми трекінгу в OpenCV

В OpenCV реалізовано декілька готових алгоритмів трекінгу. Всі вони не є надпотужними, тому працюють добре тільки на простих датасетах. Ось алгоритми, які реалізовано в OpenCV: [3]

- Boosting – алгоритм побудований на алгоритмі AdaBoost
- KCF (Kernelized Correlation Filter) – Однин із алгоритмів на основі кореляційних фільтрів
- CSRT tracker – інший алгоритм на основі кореляційних фільтрів
- MedianFlow – усереднює помилку між рухом об'єкту в кадрі t і кадрі $t+1$
- MOSSE – інший алгоритм з кореляційними фільтрами
- GoTurn – один із алгоритмів на нейронних мережах в OpenCV
- Алгоритми Лукаса-Канаде і Хорна-Шанка – алгоритми трекінгу точки на основі оптичного потоку

1.5 Оптичний потік

Оптичним потоком називається рух пікселів на зображенні. Він використовується у комп'ютерному зорі для сегментації об'єктів, але нам він цікавий тим, що завдяки ньому ми зможемо відстежувати об'єкти.

У завданні трекінгу об'єктів, найперше і найголовніше поняття є рух. У відеопотоці рух можна ідентифікувати як зміну позиції точок між кадрами.

Маємо кадри: t і $t + \delta t$, на яких деякий піксель з координатами (x, y) зміщується до координат $(x + \delta x, y + \delta y)$, тобто зміщення відбувається на $(\delta x, \delta y)$. Тоді оптичний потік буде виглядати так:

$$(u, v) = (\frac{\delta x}{\delta t}, \frac{\delta y}{\delta t})$$

Для того, щоб працювати з оптичним потоком, нам потрібно встановити деякі припущення: [4]

- При зміщенні пікселя від одного фрейма до іншого, його інтенсивність не змінюється, тобто
$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t)$$
- Піксель зміщується на незначні відстані і крок між фреймами δt є малим.

З розкладу Тейлора ми можемо отримати друге рівняння:

$$f(x + \delta x) = f(x) + \frac{df}{dx}\delta x + O(\delta x^2)$$

$$f(x + \delta x, y + \delta y, t + \delta t) \approx f(x, y, t) + \frac{df}{dx}\delta x + \frac{df}{dy}\delta y + \frac{df}{dt}\delta t$$

$$I(x + \delta x, y + \delta y, t + \delta t) \approx I(x, y, t) + \frac{dI}{dx}\delta x + \frac{dI}{dy}\delta y + \frac{dI}{dt}\delta t$$

Віднімаємо (1) від (2), поділимо на δt і отримуємо

$$I_x \frac{dx}{dt} + I_y \frac{dy}{dt} + I_t = 0, \text{або}$$

$$I_x u + I_y v + I_t = 0, \text{де}$$

$$(u, v) - \text{оптичний потік і } u, v - \text{швидкості оптичного потоку.}$$

Але це рівняння містить дві невідомі, тому однозначно не розв'язується.[4]

1.6 Алгоритм Лукаса-Канаде

Алгоритм для обчислення оптичного потоку. Припускається, що оптичний потік в дуже малому околі є однаковим для всіх точок цього околу. Нехай ми розглядаємо деякий окіл W, тоді для всіх n точок $(k, l)\ \epsilon\ W$:

$$I_x(k,l)u + I_y(k,l)v + I_t(k,l) = 0$$

Запишемо це рівняння в матричну форму:

$$\begin{bmatrix} I_x(1,1) & I_y(1,1) \\ I_x(k,l) & I_y(k,l) \\ \dots & \dots \\ I_x(n,n) & I_y(n,n) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} I_t(1,1) \\ I_t(k,l) \\ \dots \\ I_t(n,n) \end{bmatrix}$$

Отримуємо лінійну систему $Au = B$, яку можна розв'язати так: $A^T Au = A^T B$

Запишемо в матричній формі: [5]

$$\begin{bmatrix} \sum_W I_x I_x & \sum_W I_x I_y \\ \sum_W I_x I_y & \sum_W I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\sum_W I_x I_t \\ -\sum_W I_y I_x \end{bmatrix}$$

$$A^T A \qquad u \qquad A^T B$$

З цього рівняння знаходимо u

$$u = (A^T A)^{-1} A^T B$$

$$u - \text{вектор оптичного потоку.}$$

Умови для матриць $A^T A$:

- Матриця не вироджена

- Власні числа матриці не повинні бути занадто малими
- Власні числа повинні бути схожими, тобто $\frac{\lambda_1}{\lambda_2}$ не повинно бути великим. [5]

1.7 Пірамідальний алгоритм Лукаса-Канаде

Оскільки рух між фреймами може бути швидкий, то наближення Тейлора нам не підійде. Тому, замість обчислення оптичного потоку на зображенні з максимальною якістю, ми можемо зменшити його якість. Наприклад, якщо у нас є зображення N x N, томи зменшуємо його якість до $\frac{N}{2}$ x $\frac{N}{2}$, $\frac{N}{4}$ x $\frac{N}{4}$ ... Таким чином, різниця між початком і кінцем руху пікселя буде в рази меншою.

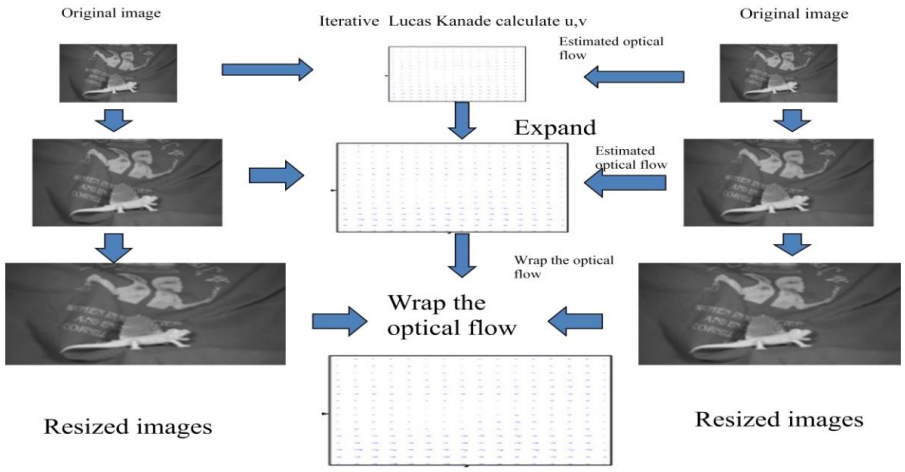


Рис. 1.3- Пірамідальний алгоритм Лукаса-Канаде [5]

1.8 Недоліки алгоритму Лукаса-Канаде

Алгоритм сам по собі не має функції «запам’ятовування» виділеної області, тобто якщо виділений об’єкт закривається іншим об’єктом, то алгоритм його губить. Для того, щоб об’єкт запам’ятовувався, використовують нейронні мережі.

1.9 Трекери на основі кореляційних фільтрів

Серед трекерів на основі кореляційних фільтрів можна виділити три найпопулярніших: MOSSE (Minimum Output Sum Of Squared Error), KCF (Kernelized Correlation Filters), ACEF (Average of Synthetic Exact Filters). Основна ідея – на основі виділеної області з цікавим нам для відстежування

об’єктом, за допомого деяких перетворень створити фільтр у вигляді матриці, який би реагував на схожу область на наступних кадрах.

Розглянемо алгоритм MOSSE. Оскільки у назві вже є підказка щодо мінімізації деякого виразу, то нам залишилось цей вираз скласти. Маємо фрейми $I_0 \dots I_n$. В першому фреймі навколо бажаного об’єкта обведемо прямокутник f_0. Для початку нам потрібний фільтр h, який буде давати хороший відгук для кожного фрейма I_p. Для простоти, будемо вважати, що ідеальний відгук буде у вигляді гауссіани:

$$G_p = 1 - \exp \frac{(x-i)^2 + (y-j)^2}{\sigma^2}$$

Тому задача мінімізації виглядатиме так:

$$D_p(i,j) = |f^p(i,j) - h|^2$$

$$h = \min_h \left|D_p(i,j) - G_p(i,j)\right|^2, p \in [0,P]$$

Тобто, ми будемо мінімізувати відхилення відгуку від бажаного відгуку. Ми отримали $P * m * n$ рівнянь з $m * n$ змінними. Далі, потрібно знайти кореляцію функцій. Для того, щоб спростити задачу, ми можемо використати розклад Фур’є :

$$F = \mathcal{F}(f)$$

$$H = \mathcal{F}(h)$$

І тоді, кореляція буде поелементним множенням в області Фур’є:

$$G = F \odot H^*, \text{де}$$

$$H^* - \text{комплексно} - \text{спряжена матриця до } H. [6]$$

Для того, щоб знайти фільтр, який ставить у відповідність вхідні дані до вихідних, алгоритм знаходить фільтр H, який мінімізує суму квадратних помилок між реальним виходом і бажаним:

$$\min_{H^*} \sum_i |F_i \odot H^* - G_i|^2$$

З цього знаходимо фільтр H^*:

$$H^* = \frac{\sum_i G_i \odot F_i^*}{\sum_i F_i \odot F_i^*}$$

На відміну від MOSSE, у ACEF трохи інший підхід до мінімізації такої помилки. Він шукає набір ідеальних фільтрів, а потім усереднює їх . Остаточна формула для фільтра ACEF дуже схожа:

$$H^* = \frac{1}{N} \sum_i \frac{G_i \odot F_i^*}{F_i \odot F_i^*}$$

Якщо навчання проходить на одному зображенні, то результати двох фільтрів співпадатимуть.

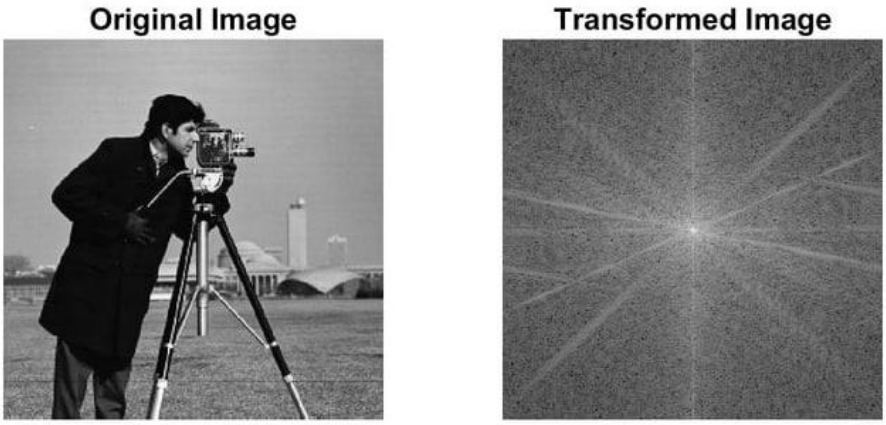


Рис. 1.4- Приклад трансформації зображення за допомогою перетворення Фур'є [20]

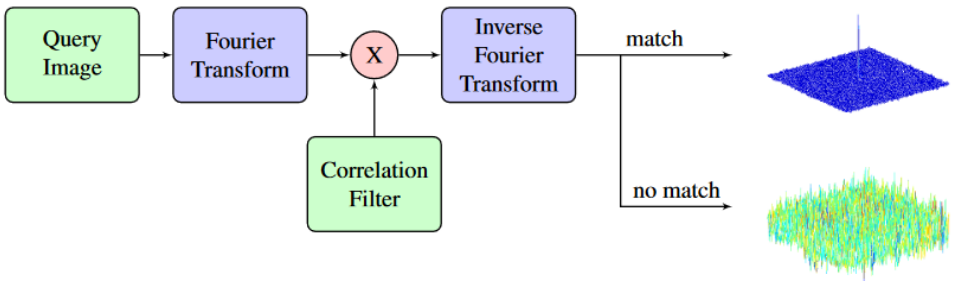


Рис. 1.5- Схема роботи алгоритмів на основі кореляційних фільтрів [21]

2. Алгоритми детекції і трекінгу об’єктів за допомогою нейронних мереж

В цьому розділі будуть представлені алгоритми на основі нейронних мереж. Ключовою складовою алгоритмів буде згорткова нейронна мережа (CNN-Convolutional Neural Network), тому варто буде сказати про неї декілька слів.

2.1 Оцінка ефективності

Для оцінки алгоритмів відстеження одного об’єкта, автори челенджу VOT (Visual Object Tracking) беруть до уваги три критерії: Accuracy- точність, Robustness- міцність (або стійкість), Expected Average Overlap (AEO). Точність – середнє перекриття між прогнозованим і реальним bbox-ом. Стійкість же визначає скільки разів трекер промахнувся впродовж роботи. EAO- оцінка середнього перекриття з довгої колекції коротких послідовностей з такими ж візуальними властивостями, що й загальний набір даних. [7]

Існує декілька метрик для оцінки ефективності алгоритмів трекінгу декількох об’єктів:

1. Multiple Object Tracking Precision (MOTP)

$$MOTP = \frac{\sum_{i,t} d_{i,t}}{\sum_t c_t}, \text{де}$$

$d_{i,t} -$ кількість правильно розпізнаних об′єктів

$c_t -$ кількість всіх розпізнавань на кадрі t

2. Multiple Object Tracking Accuracy (MOTA)

$$MOTA = 1 - \frac{\sum_t (m_t + fp_t + mme_t)}{\sum_t g_t}, \text{де}$$

$m_t -$ кількість пропущених об′єктів,

$fp_t -$ кількість $false\ positives$

$mme_t -$ кількість розбіжностей

$g_t -$ загальна кількість об′єктівна кадрі t

Проблема таких метрик в тому, що складно отримати об’єктивний результат на кастомному датасеті, бо для того, щоб отримати реальні (ground-truth) bbox-и, потрібно їх відмічати вручну на кожному фреймі, тому для таких оцінок необхідно опиратися на існуючі датасети.

2.2 CNN

CNN (Convolutional Neural Network) – найбільш популярний алгоритм для розпізнавання об’єктів. Його ідея – витягнути характеристики (feature maps) зі збірки зображень для того, щоб навчитись їх розрізняти. За це відповідають такі шари: згортковий шар (convolution layer) і пулінговий шар (pooling layer).

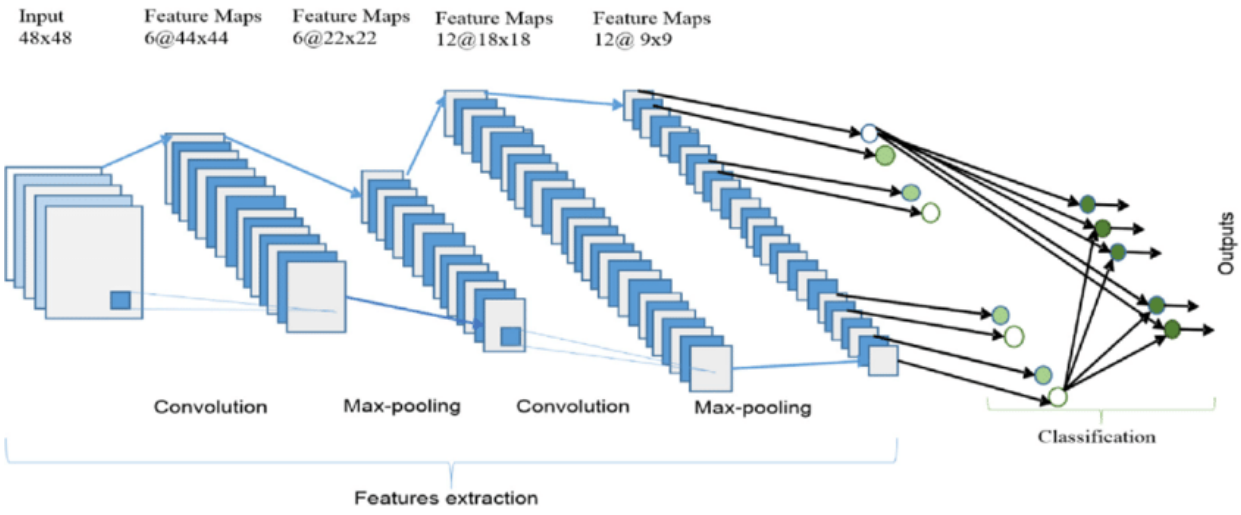


Рис. 2.1 Шари згорткової нейронної мережі [9]

Convolution layer: Для того, щоб створити feature maps, зображення у вигляді масиву прогоняються через спеціальні фільтри. Фільтри- це матриці, які відповідають за певні риси у зображенні, наприклад, прямі лінії, криві, кола.

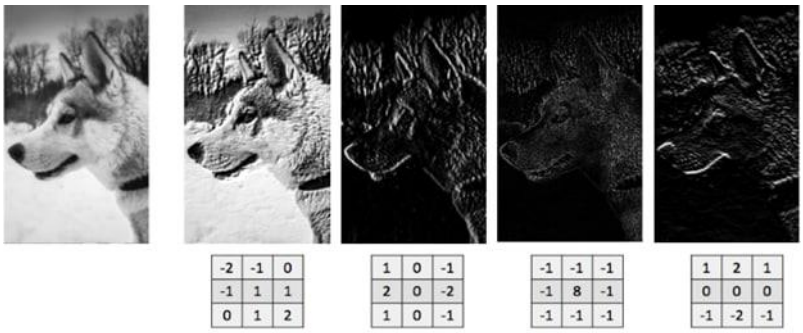


Рис. 2.2 Приклад роботи згорткового шару. [8]

Pooling layer: цей шар використовується для того, щоб зменшити розмірність зображення і одночасно зберігти важливу інформацію. Фільтри Average pooling і Max pooling– квадратні матриці, які проходять по матриці зображення і залишають або середнє, або максимальне значення.

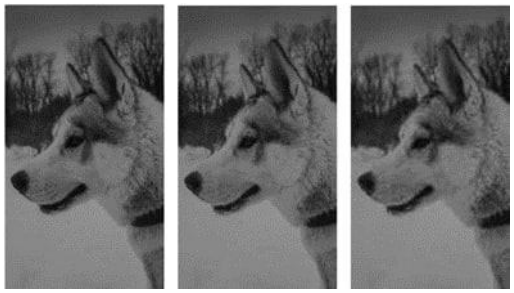


Рис. 2.3 Приклад роботи пулінгового шару. [8]

Для того, щоб передати значення карт до нейронної мережі, flattening шар перетворює матриці, що вийшли у результаті накладання convolution і pooling шарів, у вектори. Таким чином, зображення перетворюється у призначений до навчання формат.

2.2.1 Нейронні мережі для детекції об’єктів

Як ми переконалися, алгоритми для виявлення об’єктів, що працюють без навчання, погано працюють із запам’ятовуванням об’єктів, тому зараз мають популярність алгоритми, що навчені за допомогою складних нейронних мереж і, відповідно, саме вони дають найбільшу якість у знаходженні/ сегментації об’єктів, на зображеннях. Загалом, алгоритми object detection мають складну композицію із різних шарів нейронних мереж, що дозволяє

працювати з зображенням з точністю до пікселя і знаходити у ньому характеристики(features) тих об’єктів, які нам потрібні. Але чому не можна просто використати згорткові мережі? Оскільки нашою задачею є одночасно і класифікувати нам потрібний об’єкт і вказати його bounding box, ми матимемо задачу класифікації і регресії.

Ті детектори, на які варто звернути увагу- це R-CNN,Faster R-CNN, SSD і YOLO. Далі ми коротко опишемо їх і покажемо, яку із них ми будемо використовувати для відстеження об’єктів на відео.

2.2.2 R-CNN

R-CNN (Region- Based Convolutional Neural Network). ЇЇ ідея полягає в тому, щоб подати дві тисячі витягнутих регіонів із зображення (region of interest), у яких, імовірно, міститься наш об’єкт (selective search), перетворити їх у квадрати визначеного розміру і подати до згорткової нейронної мережі, щоб витягнути характеристики (features), які передаються до:

1) класифікатора для визначення чи присутній нам потрібний об’єкт, чи ні

2) регресора для визначення координат Bounding Boxes. [11]

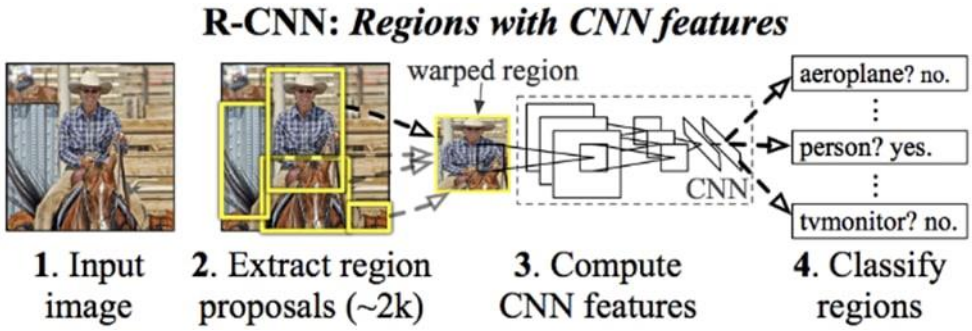


Рис. 2.4 Схема роботи R-CNN [11]

Проблема даної нейронної мережі в тому, що вона передає кожне зображення до CNN, а це займає багато часу. Тому Region-Based мережі були покращені.

2.2.3 Faster R-CNN

На відміну від її попередниці, Faster R-CNN не використовує selective search і 2000 областей. До згорткових шарів надходить зображення, на виході маємо

карти характеристик (feature maps). Ці карти передаються до блоку Region Proposal Network і до класифікатора одночасно. Region Proposal Network проходить по кожній карті характеристик і генерує гіпотези про наявність об’єкта. Наявність такого модуля значно пришвидшує час роботи нейронної мережі.

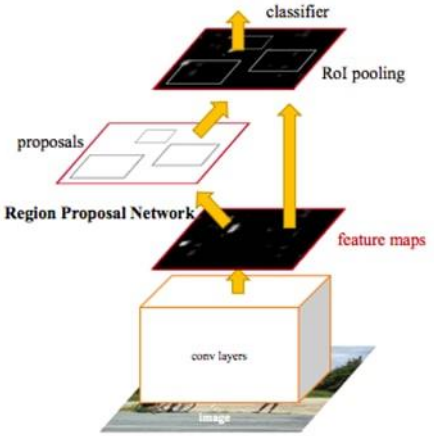


Рис. 2.5 Схема роботи R-CNN [11]

2.2.4 SSD

SSD (Single Shot Detector) – алгоритм не відноситься до region-based алгоритмів і тому згорткова мережа дивиться на зображення за один раз, а не за багато. SSD складається з двох компонентів : базова модель і голова (head) SSD. Базова модель це зазвичай попередньо натренована нейронна мережа, наприклад VGG-16, яка потрібна для того, щоб створити карти характеристик (feature maps). Голова SSD – це декілька згорткових шарів для детекції об’єктів.

2.2.5 YOLO

YOLO (You Only Look Once) – найбільш цікавий для нас алгоритм. Є аналогом SSD алгоритму, згорткова мережа застосовується відразу до всього зображення один раз. Алгоритм ділить зображення на сітку з клітин-квадратів розміру S*S. Кожна клітина намагається порахувати координати (bbox-а) відносно своїх координат і його “впевненість”. [12]

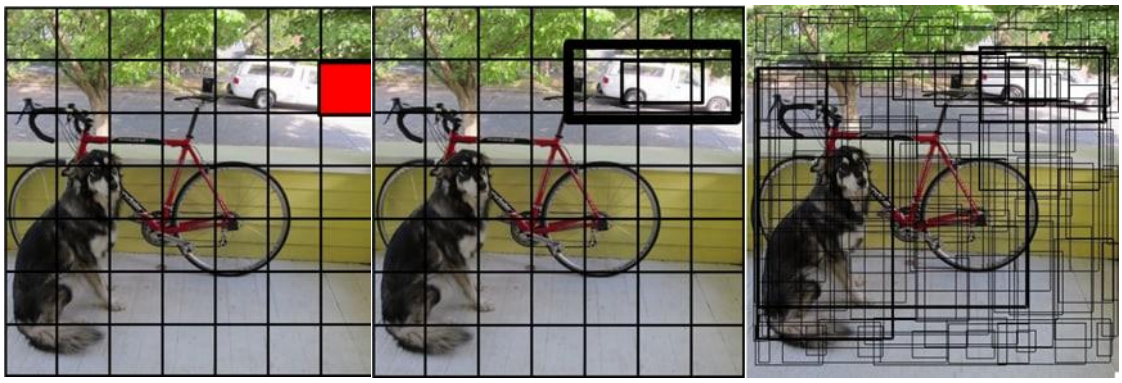


Рис. 2.6 Розділення зображення на bbox-и [12]

Оскільки багато клітин намагаються обрахувати bbox одного об’єкта, ми матимемо багато bbox-ів і різними імовірностями впевненості щодо знаходження об’єкта.

Друга частина мережі відповідає за прогнозування імовірностей класів. В кожній клітині міститься імовірність того, що якщо в даній клітині буде bbox, то імовірніше за все, він буде саме цього класу. [12]

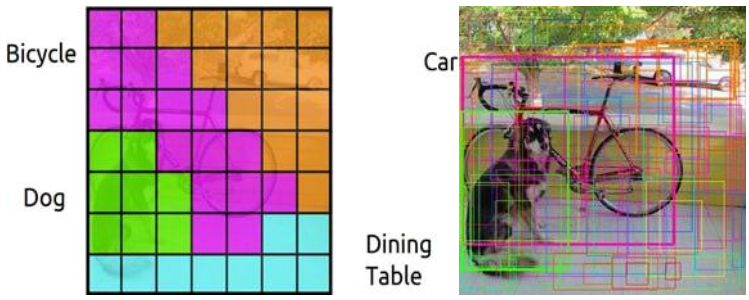


Рис. 2.7 Класифікація bbox-ів [12]

Потім ми з’єднуємо результати і матимемо багато bbox-ів різного кольору:

$$P(class) = P(class|Object) * P(Object)$$

Для того, щоб позбутися зайвих рамок, використовується Non-Maximum Supression (придушення). Цей алгоритм прибирає всі рамки з меншими показниками імовірностей.

Кожна клітина прогнозує декілька параметрів:

-координати bbox-ів

b_x, b_y, b_h, b_w, де

b_x, b_y- центр bbox-а

b_h, b_w- висота і ширина bbox-а

-значення їх “впевненості” (імовірність, що bbox містить об’єкт):

$$p_c$$

- імовірність того, що bbox містить об’єкт певного класу

$$c_{1\ldots n}$$

Тоді вектор $\vec{y}$, який згорткова мережа буде прогнозувати, можна закодувати так:

$$\vec{y} = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Метрика для вимірювання того, чи наш bbox нам підходить, чи ні, називається **Intersection over Union** (IoU), яка рахується за наступною формулою:

$$IoU = \frac{\text{Площа перетину}}{\text{Площа об}'\text{єднання}}$$

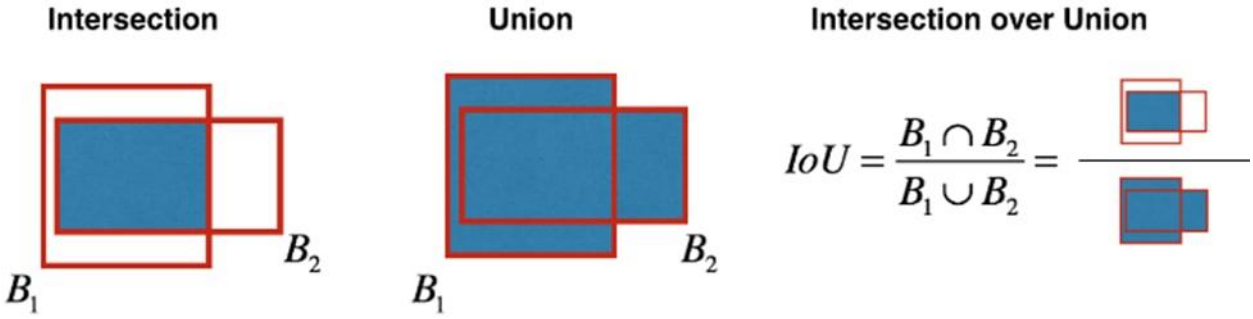


Рис. 2.8 IOU [22]

Тобто, якщо IoU між отриманим bbox-ом і ground-truth bbox-ом менший за вказаний поріг, який є гіперпараметром, тобто яке ми самі задаємо, то цей bbox ми прибираємо.

2.2.6 YOLOv3/YOLOv4

YOLOv3 і YOLOv4 останні і найбільш потужні версії YOLO. Покращення стосуються в тому числі того, що зображення розбиваються на три сітки – 13x13, 26x26 і 52x52, тому алгоритм взмозі розпізнати як малі, так і великі об’єкти.

Але нам він цікавий є тим, що він швидший за R-CNN майже у 100 разів, що дозволяє проводити локалізацію об’єктів без затримок, онлайн. [10]

2.3 Від зображення до відео

Результат роботи алгоритму: bbox навколо об’єкту і впевненість, що це саме цей об’єкт. Якщо ми працюємо з відео, то алгоритм розбиває його на фрейми і працює з кожним фреймом окремо. Таким чином, кожен об’єкт на новому фреймі має новий bbox, незалежний від попереднього. Тому виникає питання- як зробити так, щоб інформація про переміщення об’єкта (і, відповідно, bbox-а) переходила від фрейма до фрейма, щоб ми змогли відслідкувати траєкторію.

Перша очевидна проблема – пов’язати минулий bbox об’єкта з наступним. Інтуїтивне рішення – якщо різниця між центрами попереднього bbox-а і наступного найменша між усіх інших bbox-ів, то ми можемо сприймати наступний bbox за “правомірного наступника”.

Друга проблема такого завдання – перетин об’єктів. Нашою задачею є зробити так, щоб після перетину двох об’єктів з bbox-ами, вони не загубились.

2.3.1 DeepSort

Головна особливість цього алгоритму полягає в тому, що він вміє запам’ятовувати об’єкти, зображені в bbox-ах. Як це відбувається? Після процедури детекції, об’єкти, які були знайдені в bbox-ах, проганяються через згорткову мережу і проходять feature extraction. Тобто, кожне зображення отримує свій вектор характеристик, який допомагатиме відслідковувати той bounding box, в якому він знаходиться.

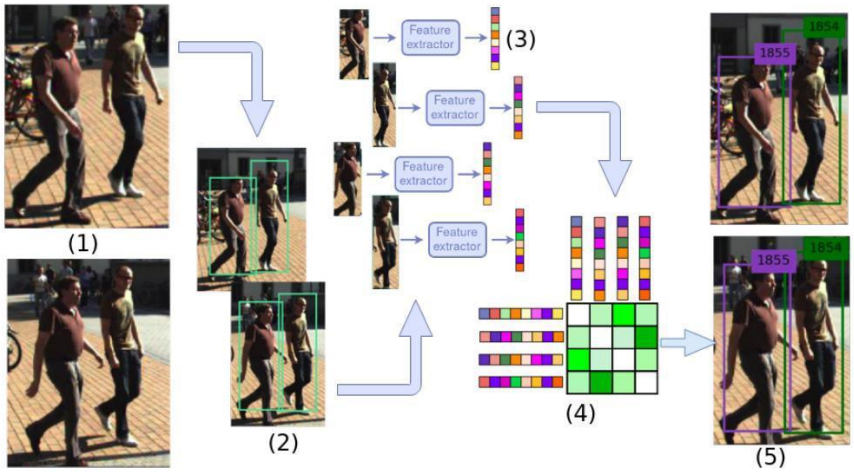


Рис. 2.9 Приклад генерування вектора характеристик [13]

2.3.2 Фільтр Калмана

У задачах, де показники вимірюються деякими датчиками або сенсорами, ці інструменти мають похибку, тому неможливо визначити істинне значення. Фільтр Калмана допомагає згладити цю похибку так, щоб обчислене значення наближалося до оптимального. Сам алгоритм поділяється на два етапи: прогнозування та корекція. Під час етапу прогнозування ми отримуємо значення від сенсора із похибкою, а під час корекції намагаємося зменшити її [15]. Нехай стан об’єкту змінюється за формулою

$$\hat{x}_t = A\hat{x}_{t-1} + Bu_t \,,$$

$$\widehat{P}_t = AP_{t-1}A^T + Q \,,$$

де A, B і Q – матриці, що відповідають за рух динамічної системи, P – коваріаційна матриця помилок

Етап корекції визначається тим, що нам необхідно розставити вагові коефіцієнти між тим значенням, що порахував сенсор і між оптимальним (оціненим за допомогою фільтра Калмана на минулому кроці) [15]. Цей ваговий коефіцієнт називається посилення Калмана (Kalman gain), який визначається за формулою:

$$K_t = \frac{\widehat{P}_t H^T}{\left(H\widehat{P}_t H^T + R\right)}$$

$$0 < K_t < 1$$

$$H - \text{матриця переходу стану. зазвичай є } 1$$

Після того, як ми обчислили посилення Калмана, можемо поновити оцінки з урахуванням вимірів, зроблених в поточний момент часу:

$$x_t = \hat{x}_t + K_t(z_t - H\hat{x}_t)$$

І також поновити матрицю помилок

$$P_t = (I - K_t H)\widehat{P}_t$$

Фільтр Калмана є незамінним алгоритмом для трекінгу об’єктів. Для алгоритму DeepSort початковий стан є вектором з восьми змінних:

$$\bar{X} = (u, v, a, h, u', v', a', h'), \text{де}$$
$$(u, v) - \text{центр } bbox - \text{а}$$
$$a - \text{співвідношення сторін}$$
$$h - \text{висота } bbox - \text{а}$$
$$u', v', a', h' - \text{швидкості вищевказаних змінних}$$

Етап прогнозування: при зміщенні цілі, беруться координати цілі на попередньому кадрі і його швидкість, які використовуються для прогнозування положення цілі на поточному кадрі.

Етап корекції: пораховані значення під час попереднього етапу і поточне спостережуване значення зважуються за допомогою посилення Калмана і таким чином ми маємо прогнози для наявних bbox-ів.

Після того, як ми маємо bbox на кадрі A, його прогноз по фільтру Калмана, і новий bbox на кадрі B, то, ми маємо завдання поєднати їх. Для вирішення цього завдання, нам необхідна метрика відстані для bbox-ів і алгоритм для зіставлення прогнозованої траєкторії з bbox-ом з кадру B. Такими будуть метрика з поєднання відстані Махаланобіса і косинусної всідстані і угорський алгоритм.

Відстань між двома bbox-ами рахується наступним чином. Зображення з обох bbox-ів прогоняється через згорткову мережу, з якої видалений шар класифікації. Тобто, на виході з цієї мережі ми отримуємо вектори ознак з обох зображень розмірністю 128x1.

2.3.3 Відстань Махаланобіса

Це спосіб обчислення схожості двох вибірок. [14]

$$d(\vec{x},\vec{y}) = \sqrt{(\vec{x}-\vec{y})^T S^{-1}(\vec{x}-\vec{y})} \text{ де } S - \text{матриця коваріації.}$$

2.3.4 Косинусна Відстань

$$1 - \cos(\theta) = \frac{\vec{x} * \vec{y}}{\left|\left|\vec{x}\right|\right| * ||\vec{y}||}$$

Таким чином, відстань між bbox-ом, що був знайдений YOLO і таким, що був знайдений фільтром Калмана (Deep Assosiation Metric), буде така: [13]

$$D = Lambda * D_k + (1 - Lambda)D_a, \text{де}$$

$$D_a - \text{косинусна відстань}$$

$$D_k - \text{відстань Махаланобіса}$$

2.3.5 Угорський алгоритм

Це оптимізаційной алгоритм, який ефективно вирішює задачі асоціації. Нам він потрібен для того, щоб для n штук bbox-ів присвоїти n траєкторій. [14]

Виконується задача мінімізації:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \rightarrow (\min)$$

3. Реалізація різних типів алгоритмів трекінгу

У цьому розділі ми зобразимо, доповнимо і порівняємо алгоритми з розділів 1 і 2, спробуємо їх порівняти там, де це можливо.

3.1 Опис задачі

Метою роботи є порівняти різні типи алгоритмів трекінгу на різних відео. Оскільки більшість із розглянутих вище алгоритмів є чутливими до руху камери, то буде справедливо робити висновки щодо їх порівняння тільки опираючись на відео з статичних камер.

3.2 Реалізація алгоритму Лукаса-Канаде

Алгоритм Лукаса-Канаде вже є реалізований в бібліотеці OpenCV. Варто нагадати, що він рахує оптичний потік, що означає- ми можемо стежити за бажаною точкою. Пірамідальний алгоритм Лукаса-Канаде реалізовано за допомогою функції $cv2.calcOpticalFlowPyrLK()$. Вона приймає такі аргументи:

$prevImg -$ перше (попереднє) зображення

$nextImg -$ наступне зображення

$prevPts -$ точка, для якої повинен бути порахований оптичний потік

$nextPts -$ наступні точки (зазвичай $None$)

$winSize -$ величина околу

$maxLevel -$ кількість рівнів пірамід

$criteria -$ параметри для ітеративного пошуку

Першим кроком необхідно обрати точку з областю для спостереження. Для цього використаємо функцію cv2.$selectROI()$, яка нам дозволить вибрати прямокутник на першому зображенні відео, і ми з цього прямокутника виберемо центр, що і буде нашою точкою. Надалі ми розглядатимемо два типи відео- з рухомою камерою і нерухомою .

Для кожного наступного фрейма з’являється одна нова точка, тому для того, щоб створити траєкторію, всі попередньо знайдені точки повинні зберігатись у новому масиві. Останній крок - порахувати оптичний потік за допомогою

алгоритму. Результат роботи алгоритму – на відео з’являються точки, які і будуть траєкторією руху відстежуваної точки:

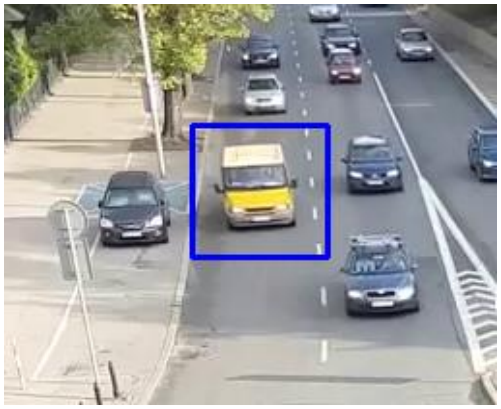


Рис.3.1 Вибір ROI. Оригінал-[23]

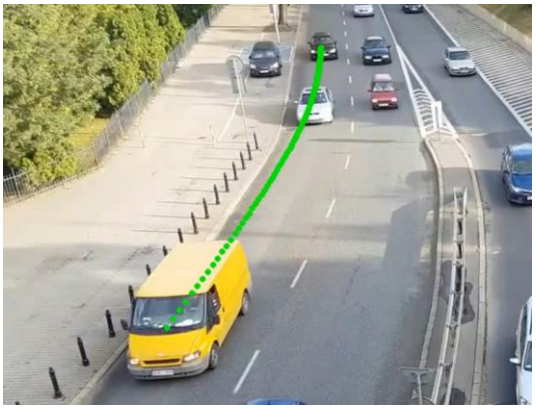


Рис.3.2 Результат роботи LK. Оригінал- [23]

Рис. 3.3 Вибір ROI іншого відео. Оригінал – [24]

Рис. 3.4 Результат роботи LK Оригінал – [24]

Але, не завжди алгоритм працює так гладко. Алгоритм провалюється коли спостережувана точка перетинається з іншим об’єктом, коли камера рухається, або коли об’єкт зникає на деякий час із поля зору:

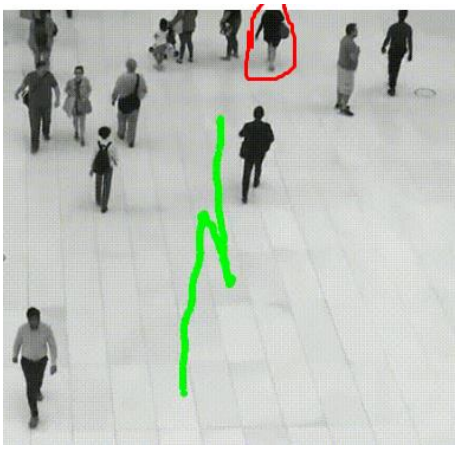


Рис. 3.5 Алгоритм LK втрачає ціль. Оригінал – [24]

Варто зазначити, що оскільки даний алгоритм відповідає тільки за рух точки, то обмежувальна рамка тут не має сенсу, тому якщо її тут робити, то вона буде тут суто символічним прямокутником навколо точки, який не матиме з розмірами об'єкта нічого спільного. Таким прямокутником можна зробити прямокутник з розміром вікна роботи алгоритму, який зазначається в параметрі $winSize$ функції $cv2.calcOpticalFlowPyrLK()$.

3.3 Доповнення до алгоритму Лукаса-Канаде

Також можна спробувати інший спосіб з цим алгоритмом. Виберемо *ROI* навколо бажаного об'єкту і згенеруємо всередині неї n (наприклад 100) точок. І прослідкуємо за допомогою алгоритму Лукаса-Канаде за всіма такими точками. Також для зручності можна обвести b-box навколо таких точок на кожному фреймі.

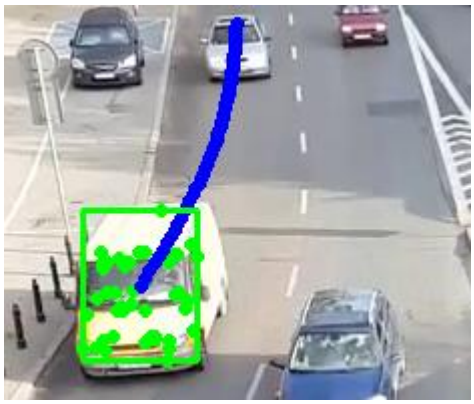


Рис.3.6 Алгоритм слідкує за багатьма незалежними точками. Оригінал-[23]

Оскільки деякі точки будуть ламати алгоритм, тобто рахувати потік з великими похибками, bbox буде обводити також і ті точки, які “втекли” дуже далеко і не мають нічого спільного з відстежуваною областю, як точка справа на Рис. 3.7:

Рис. 3.7. Точка, що відхилилася, повинна бути видалена

І таким чином ми отримуємо завдання видалити такі точки. Застосуємо припущення: більшість точок відстежуються правильно, тому такі точки утворюватимуть область з найбільшою щільністю. Для вирішення такої задачі можна застосувати алгоритм кластеризації DBSCAN. Ми отримаємо кластери, які засновані на щільності, тобто алгоритм розіб’є наші точки таким чином, що ми зможемо відсортувати кластери за кількістю точок в кластері. Область з найбільшою щільністю точок ми збережемо, а інші кластери видалимо.

Єдиний недолік такогодоповнення – швидкість. Для роботи в форматі онлайн він буде надзвичайно повільним і буде обчислюватися приблизно зі швидкістю 2 fps.

3.4 Реалізація алгоритму MOSSE

Алгоритм реалізовано в бібліотеці OpenCV. Ми скористаємося її інтерфейсом. Спочатку обирається ROI за допомогою функції cv2.$selectROI()$. Далі йде ініціалізація алгоритму $cv2.TrackerMOSSE\_create()$ і за допомогою цикла, на кожному фреймі відстежується bbox: $tracker.update(frame)$.

Алгоритм має деякі переваги перед алгоритмом Лукаса-Канаде, а саме: сильні коливання об’єкта не гублять його bbox, невеликі перетини з іншими об’єкта також працюють стабільніше.

Рис. 3.8 Вибір ROI.
Оригінал-[23].

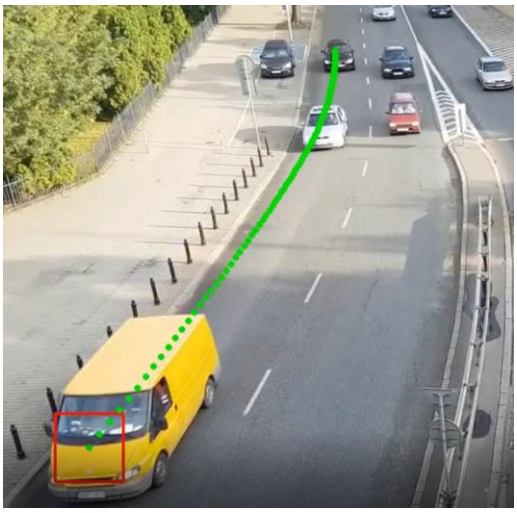


Рис. 3.9 Робота алгоритму MOSSE.
Оригінал-[23]

На простому відео алгоритм працює добре, проте при зміні масштаба об’єкта, його bbox не змінюється. Проте, при окклюзії, алгоритм губиться і перестає відслідковувати відповідний bbox.

Рис. 3.10 Вибір ROI для іншого відео
Оригінал - [25].

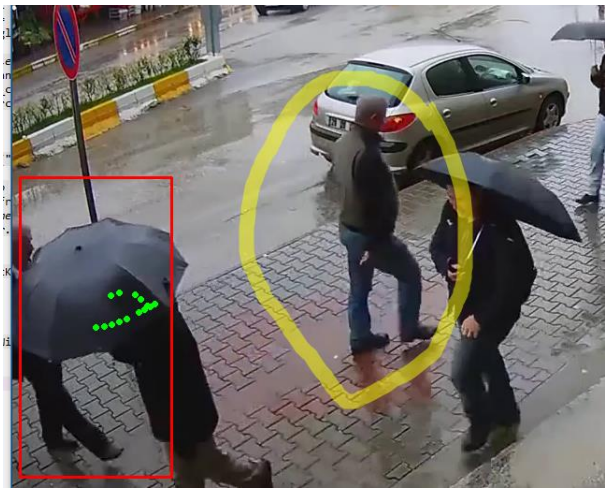


Рис. 3.11 MOSSE втрачає ціль
Оригінал - [25].

Тут бачимо, що алгоритм загубив ціль.

3.5 Реалізація YOLO

Код алгоритму взятий з [1]

Алгоритм працює відмінно в порівнянні з попередніми алгоритмами на переднавчених класах і серед таких класів є автомобіль і людина.

Загалом алгоритм видає всі детекції для всіх об’єктів, що були знайдені на відео, але в нашому випадку нам буде цікаво відстежити траєкторію раніше спостережуваного об’єкта.

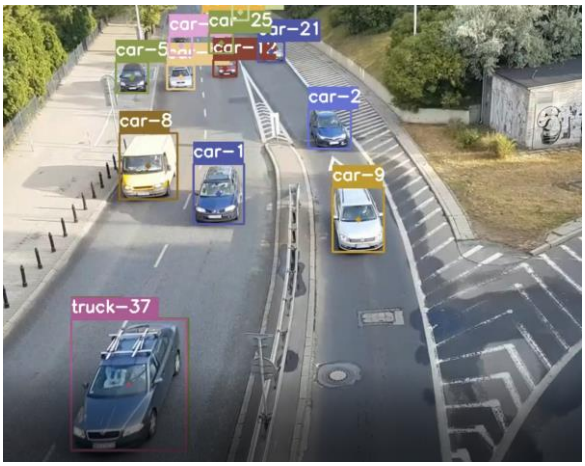


Рис. 3.12 Робота алгоритму DeepSort. Оригінал - [23].

Але з всього потоку можна вибрати жовтий автомобіль. Я це зробив наступним чином: завантажив всі bbox-и на кожному фреймі у вигляді датасету. Вибрав один нам потрібний клас за номером, який був визначений алгоритмом (у нашому випадку це клас 8) і зобразив тільки його і траєкторію його руху:

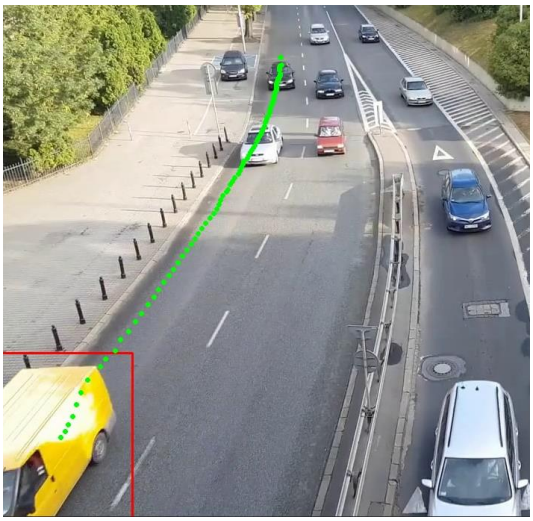


Рис. 3.13 Траєкторія руху за допомогою DeepSort. Оригінал- [23]

Також алгоритм запам’ятовує риси кожного об’єкта і тому він не губиться, коли об’єкт затуляється іншим, або коли зникає з кадру:

Рис. 3.14 Об'єкт зникає з поля зору Оригінал – [25]

Рис. 3.15 DeepSort знаходить загублений об'єкт. Оригінал – [25]

На цьому фото можна побачити як об'єкт затуляється іншим об'єктом, але потім алгоритм все одно його знаходить і відстежує. Траєкторія намальована заздалегідь, бо вже була знайдена алгоритмом.

3.6 Порівняння алгоритмів

На простих відео всі алгоритми показують схожі траєкторії. Проте, при будь-яких ускладненнях, як зникнення об'єкту з видимості, алгоритми без навчання ламаються.

Траєкторії руху автомобіля на графіку:

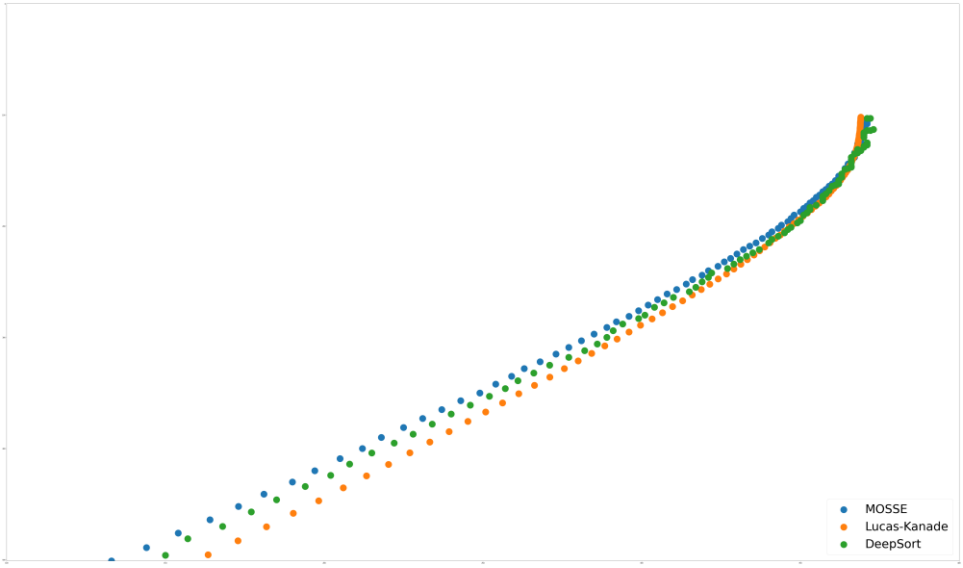


Рис. 3.16 – Траєкторії різних алгоритмів на графіку

Таким чином, незважаючи на те, що алгоритмів трекінгу без навчання використовують великий спектр математичних методів, вони поки не здатні запам’ятовувати об’єкт, і тоді робота алгоритму втрачає сенс, бо «стерильні» відео , де спостережувані об’єкти не перетинаються або не зникають, є доволі рідкісним явищем. До таких складнощів можна додати зміну масштабу об’єкта, шуми і ротацію камери, і тоді такі алгоритми вимагатимуть значних вдосконалень.

Висновки

Під час виконання роботи були проаналізовані різні за походженням алгоритми трекінгу. Також ми оцінили їх роботу на відеопотоках з нерухомими камерами і також порівняли траєкторії руху, які вони спрогнозували.

У результаті прийшли до висновку – алгоритми на нейронних мережах є універсальними, бо окрім того, що вони відстежують об’єкт, вони можуть зберігати його характеристики так, щоб об’єкт не губився під час накладання іншого об’єкта або під час зникнення з поля зору камери.

Алгоритм MOSSE. Із недоліків: при приближенні об’єкта до камери, bbox об’єкта не змінює свої розміри, тому в такому випадку bbox обмежуватиме тільки частину об’єкта. Також ми побачили, що алгоритм погано запам’ятовує відстежуваний об’єкт. Із переваг: На відео, в яких об’єкт у високій якості і не перетинається з іншими, алгоритм працює відмінно.

Алгоритм Лукаса- Канаде. Із недоліків: відстежує точку з околом, а не об’єкт, що спричиняє швидку втрату спостережуваної точки при оклюзії. Із переваг – швидке обчислення, точне обчислення при нешвидких рухах точки.

Алгоритм DeepSort. Із недоліків: нелінійна швидкість, відстеження всіх об’єктів відразу, а не бажаного, що уповільнює роботу. Із переваг: Є можливість запам’ятомування об’єкта, що дозволяє відслідковувати його протягом всього відео, навіть після того, як об’єкт зникне з поля зору.

Перелік джерел

[1] https://nanonets.com/blog/object-tracking-deepsort

[2] https://docs.opencv.org/4.x/

[3] https://learnopencv.com/object-tracking-using-opencv-cpp-python/

[4] Horn, Berthold & Schunck, Brian. (1981). Determining Optical Flow. Artificial Intelligence. 17. 185-203. 10.1016/0004-3702(81)90024-2.

[5] Sharmin, Nusrat & Brad, Remus. (2012). Optimal Filter Estimation for Lucas-Kanade Optical Flow. Sensors (Basel, Switzerland). 12. 12694-709. 10.3390/s120912694.

[6] Bolme, David & Beveridge, J. & Draper, Bruce & Lui, Yui. (2010). Visual object tracking using adaptive correlation filters. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2544-2550. 10.1109/CVPR.2010.5539960.

[7] Bernardin, Keni & Stiefelhagen, Rainer. (2008). Evaluating multiple object tracking performance: The CLEAR MOT metrics. EURASIP Journal on Image and Video Processing. 2008. 10.1155/2008/246309.

[8] https://morioh.com/p/c944e8e7ee9f

[9] Alom, Md. Zahangir & Taha, Tarek & Yakopcic, Chris & Westberg, Stefan & Sidike, Paheding & Nasrin, Mst & Hasan, Mahmudul & Essen, Brian & Awwal, Abdul & Asari, Vijayan. (2019). A State-of-the-Art Survey on Deep Learning Theory and Architectures. Electronics. 8. 292. 10.3390/electronics8030292.

[10] https://pjreddie.com/darknet/yolo/

[11] https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[12] arXiv:1506.02640

[13] https://russianblogs.com/article/68291549502/

[14] arXiv:1703.07402

[15] https://habr.com/ru/post/140274/

Джерела зображень і відео

[16] https://books.google.com/ngrams

[17] Zou, Zhengxia & Shi, Zhenwei & Guo, Yuhong. (2019). Object Detection in 20 Years: A Survey.

[18] https://medium.com/@vsreedharachari/practical-implementation-of-object-detection-on-video-with-opencv-and-yolo-v3-pre-trained-weights-a2d2995aac41

[19] https://aidetic.in/blog/2020/10/05/object-tracking-in-videos-introduction-and-common-techniques/

[20] https://www.projectrhea.org/rhea/index.php/The_2-D_Fourier_Transform_and_Images

[21] http://hal.cse.msu.edu/papers/mmcf/

[22] https://habr.com/ru/post/514450/

[23] https://www.youtube.com/watch?v=MNn9qKG2UFI

[24] https://www.youtube.com/watch?v=WvhYuDvH17I

[25] https://www.youtube.com/watch?v=GJNjaRJWVP8