МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Оксана ШОВКОПЛЯС
(підпис)
грудня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»
освітньо-професійної програми «Інформатика»
на тему: Інформаційна технологія адаптивного відстеження об'єктів на базі гейтових модулів
здобувача групи ІН.м-33 Арбузов Владислав Тимофійович

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Владислав АРБУЗОВ
(підпис)

Керівник Артем Коробов
старший викладач

(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Оксана ШОВКОПЛЯС
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 «Комп'ютерні науки», освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-33 Арбузов Владислав Тимофійович

1. Тема роботи: Інформаційна технологія адаптивного відстеження об'єктів на базі гейтових модулів
затверджую наказом по СумДУ від *«03» листопада 2024 року № 1257-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 06 грудня 2024 року*
3. Вхідні дані до кваліфікаційної роботи: Розробка інформаційної технології адаптивного трекінгу об'єктів на базі гейтових модулів.
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2)Огляд технологій використання Gumbel-Softmax та ключові бібліотеки.3) Розробка системи використовуючи гейтовий модуль. 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____

6. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	20.08.24 – 01.09.24	
2	*Огляд технологій використання Gumbel-Softmax та ключові бібліотеки*	02.09.24 – 10.10.24	
3	*Розробка системи використовуючи гейтовий модуль*	11.10.24 – 29.10.24	
4	*Аналіз отриманих результатів*	29.10.24 – 30.10.24	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	01.11.24 – 30.11.24	

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 60 стр., 6 рис., 5 таб., 22 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв'язанню важливої практичної задачі відстеження об'єктів у реальному часі, що має застосування в системах відеоспостереження, автономного транспорту, дронів, робототехніки та інших галузях. Впровадження адаптивних технологій, зокрема гейтових модулів, дозволяє оптимізувати використання ресурсів і підвищити ефективність трекінгових систем.

Об'єкт дослідження - процес адаптивного відстеження об'єктів

Предмет дослідження - технологія адаптивного вибору компонентів трекера на основі гейтових модулів із використанням Gumbel-Softmax.

Мета роботи — розробка інформаційної технології адаптивного відстеження об'єктів на основі гейтових модулів, що забезпечує вибір компонентів трекера залежно від умов із мінімальними витратами ресурсів.

Методи дослідження — методи комп'ютерного зору, технологія Gumbel-Softmax для вибору компонентів, алгоритми відстеження об'єктів у реальному часі, а також інструменти для аналізу та тестування.

Результати — розроблено адаптивний трекер із використанням гейтових модулів на базі NanoTrack, інтегровано Gumbel-Softmax для вибору шляхів у моделі. Проведено експериментальну оцінку точності, швидкості роботи та адаптивності трекера на наборах даних для трекінгу (наприклад, OTB, VOT). Показано, що модифікація забезпечує зменшення обчислювальних витрат із мінімальною втратою точності. Надано графіки, що демонструють залежність точності від параметрів системи.

Ключові слова: АДАПТИВНИЙ ТРЕКЕР, ВІДСТЕЖЕННЯ ОБ’ЄКТІВ, GUMBEL-SOFTMAX, ГЕЙТОВІ МОДУЛІ, PYTORCH, НАНОТРЕКЕР.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми роботи

Відстеження об’єктів у реальному часі є важливою задачею сучасного комп’ютерного зору, що знаходить застосування у таких галузях, як відеоспостереження, робототехніка, автономний транспорт, доповнена реальність тощо. Використання глибоких нейронних мереж дозволило досягти високої точності трекінгу, однак така точність часто супроводжується значними обчислювальними витратами. Це створює перешкоди для впровадження трекерів на пристроях із обмеженими ресурсами, таких як дрони чи мобільні пристрої.

Тема роботи пов’язана з використанням гейтових модулів, інтегрованих із методом Gumbel-Softmax, що дозволяє динамічно адаптувати роботу трекера до конкретних умов. Такий підхід спрямований на зменшення обчислювальних витрат із мінімальною втратою точності. Розроблена технологія може бути використана в різних системах реального часу, що підкреслює практичну значущість теми роботи.

Актуальність

Тема роботи є актуальною, оскільки вона присвячена розробці інформаційної технології, яка дозволяє адаптивно обирати компоненти моделі трекера залежно від умов. Це особливо важливо для систем реального часу, де обчислювальні ресурси обмежені. Інтеграція гейтових модулів із Gumbel-Softmax забезпечує баланс між продуктивністю та точністю, що є ключовою задачею у сфері комп’ютерного зору.

Об’єкт дослідження: Процес адаптивного відстеження об’єктів у реальному часі.

Предмет дослідження: Методи та інструменти, що дозволяють реалізувати адаптивний вибір компонентів трекера на основі гейтових модулів із використанням Gumbel-Softmax.

Новизна

Новизна роботи полягає в інтеграції методу Gumbel-Softmax у процес трекінгу об’єктів для адаптивного вибору компонентів трекера. Запропонований підхід дозволяє:

- Динамічно активувати або деактивувати певні шляхи в архітектурі моделі.
- Знизити витрати ресурсів при мінімальній втраті точності.
- Оптимізувати роботу трекера в умовах реального часу.

Проведено експериментальну оцінку інтеграції Gumbel-Softmax у трекер NanoTrack, результати якої підтвердили ефективність запропонованого підходу.

Структура. Дана робота складається зі вступу, основних розділів, висновків, списку використаних джерел та додатків.

- У вступі описано обґрунтування вибору теми, актуальність дослідження, мету, завдання, новизну та структуру роботи.
- У першому розділі (аналітичний огляд) наведено огляд сучасного стану задач трекінгу, аналіз існуючих рішень і постановку задачі.
- У другому розділі розглянуто принцип роботи Gumbel-Softmax, описано використання гейтових модулів та інтеграцію в трекер NanoTrack.
- У третьому розділі (експериментальна оцінка) представлено методологію тестування, аналіз точності та продуктивності модифікованого трекера. Також наведено графіки та інші візуалізації.
- У висновках підсумовано результати роботи, зазначено обмеження та перспективи подальших досліджень.

- У додатках наведено фрагменти коду.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Сучасний стан задач трекінгу

Трекінг об'єктів у відеопотоці є однією з ключових задач комп'ютерного зору, яка включає процес локалізації об'єкта на кожному кадрі відео та відстеження його руху протягом усього відеопотоку. Ця задача залишається актуальною через широкий спектр її застосувань, включаючи відеоспостереження, автономний транспорт, робототехніку, доповнену реальність, спортивну аналітику, автоматизацію промислових процесів тощо.

Основна складність трекінгу полягає в необхідності підтримувати точність і стабільність відстеження навіть у складних умовах, таких як:

- Часткове або повне перекриття об'єкта іншими об'єктами.
- Зміна масштабу, ракурсу або освітлення.
- Поява шумів у відео чи швидких змін у траєкторії об'єкта.
- Одночасне відстеження кількох об'єктів у кадрі.

У сучасних системах трекінгу значну роль відіграють глибокі нейронні мережі (DNN), які забезпечують високий рівень точності завдяки здатності аналізувати складні особливості об'єктів. Проте їх використання часто обмежується великими обчислювальними витратами, що стає проблемою для реальних додатків із низькими ресурсами, таких як дрони, портативні пристрої чи вбудовані системи.

Ключові вимоги до трекінгових систем:

1. **Точність**:
    - Система має визначати положення об'єкта з високою просторовою точністю.
2. **Швидкість**:

- Реалізація повинна працювати в режимі реального часу (зазвичай 30 FPS і вище).

3. **Стійкість**:
   - Відстеження має бути стійким до змін у середовищі (наприклад, перекриття, зміна ракурсу).
4. **Обчислювальна ефективність**:
   - Особливо важлива для пристроїв із обмеженими ресурсами.

Категорії задач трекінгу

Сучасні системи трекінгу можна умовно розділити на три основні категорії:

1. **Однооб'єктний трекінг**:
   - Відстежується лише один об'єкт у кадрі.
   - Завдання: зберігати його позицію навіть при появі перешкод.
2. **Багатооб'єктний трекінг (MOT)**:
   - Завдання: одночасно відстежувати кілька об'єктів, навіть якщо вони мають подібний вигляд.
   - Ускладнюється через необхідність ідентифікації кожного об'єкта.[1]
3. **Трекінг із передбаченням (predictive tracking)**:
   - Поєднання трекінгу з передбаченням траєкторії об'єкта на основі його минулих положень.

1.2 Аналіз аналогічних проєктів

Сучасні методи трекінгу об'єктів базуються на різних підходах, що розвивалися від класичних алгоритмів до сучасних глибоких нейронних мереж. Кожен із цих підходів має свої переваги й обмеження залежно від сфери застосування та умов роботи.

1.2.1 Класичні методи

До появи глибокого навчання більшість трекерів базувалися на класичних алгоритмах, таких як фільтри Калмана та методи кореляції. Ці методи мали просту реалізацію і працювали ефективно в обмежених умовах.

Фільтри Калмана

Фільтри Калмана є одним із найпоширеніших підходів у класичних системах трекінгу. Вони використовують попередні положення об’єкта для передбачення його майбутньої траєкторії. Це робиться за допомогою математичної моделі, яка враховує швидкість, прискорення та інші параметри руху.

Переваги:

- Низькі обчислювальні витрати.
- Простота реалізації.
- Ефективність для об'єктів із передбачуваними траєкторіями.

Недоліки:

- Погана стійкість до змін вигляду об'єкта.
- Нестабільна робота за умов перекриття чи різких змін у русі.

Методи на основі кореляції

Методи кореляції аналізують подібність між регіонами зображення для пошуку об’єкта. Вони працюють за принципом зіставлення шаблона об’єкта з частинами кадру.

Переваги:

- Простота і швидкість реалізації.
- Ефективність для статичних об’єктів.

Недоліки:

- Низька точність при зміні масштабу, освітлення чи ракурсу об’єкта.
- Нестабільність за наявності шуму або перекриттів.

1.2.2 Глибокі нейронні мережі

Сучасні моделі трекінгу базуються на глибокому навчанні, що дозволяє трекерам обробляти складні сценарії і враховувати багато факторів, таких як змінний вигляд об'єкта чи динамічне оточення. Ці підходи забезпечують значно вищу точність порівняно з класичними методами.

Siamese Networks

Siamese Networks працюють за принципом порівняння шаблона об'єкта з регіонами в кадрі. Вони використовують дві гілки нейронної мережі (одна для шаблона, інша для поточного кадру), які навчаються спільному простору ознак.[4]

Приклади:

- **SiamFC (Fully Convolutional)**: перший трекер на базі Siamese Networks.
- **SiamRPN (Region Proposal Network)**: включає механізм пропозицій регіонів для точнішого визначення положення об'єкта.

- **NanoTrack**: полегшена версія SiamRPN, оптимізована для пристроїв із низькими ресурсами.

Переваги:

- Висока швидкість роботи.
- Простота реалізації.
- Сумісність із режимом реального часу.

Недоліки:

- Неадаптивність до змінних умов (наприклад, перекриттів чи шуму).
- Високі обчислювальні витрати в складних сценаріях.

Transformer-based трекери

Transformer-базовані трекери використовують механізми уваги для врахування глобального контексту, що дозволяє ефективно обробляти складні сцени.

Приклад: TransT, що поєднує переваги Transformers і класичних методів трекінгу.

Переваги:

- Здатність враховувати довгострокові залежності.
- Висока точність навіть у складних умовах.

Недоліки:

- Значні обчислювальні витрати.
- Складність реалізації.

1.2.3 Адаптивні методи

Адаптивні підходи є новим напрямком у трекінгу об'єктів. Вони включають динамічні механізми вибору компонентів або шарів моделі залежно від контексту.

Гейтові модулі

Гейтові модулі дозволяють активувати або деактивувати певні шляхи чи шари в моделі залежно від вхідних даних.

Переваги:

- Зниження обчислювальних витрат.
- Покращення продуктивності в реальних умовах із різними вимогами.

Недоліки:

- Потреба у складному налаштуванні.

Gumbel-Softmax

Gumbel-Softmax використовується для диференційованого вибору шляхів, що дозволяє інтегрувати цей механізм у градієнтний спуск.

Переваги:

- Ефективний баланс між точністю та швидкістю.
- Сумісність із існуючими нейронними архітектурами.

Недоліки:

- Залежність від налаштування температури `tau`.

Таблиця 1.1 ілюструє порівняння різних методів, які використовуються для трекінгу об'єктів у відеопотоці. Вона включає класичні методи, сучасні підходи, засновані на глибокому навчанні, та адаптивні методи. Кожна модель має свої унікальні особливості, що визначають її ефективність у різних сценаріях.

Таблиця 1.1 Порівняння методів для трекінгу

Методи	Опис	Переваги	Недоліки
Фільтри Калмана	Алгоритм для передбачення траєкторії об'єкта на основі попередніх станів.	- Низькі обчислювальні витрати. - Простота реалізації.	- Погано працює при різких змінах руху. - Не враховує зміну вигляду об'єкта.
Методи кореляції	Зіставляють шаблон об'єкта з регіонами в кадрі.	- Простота реалізації. - Ефективність для статичних об'єктів.	- Низька точність при зміні масштабу чи ракурсу. - Чутливість до шуму.
Siamese Networks	Глибокі нейронні мережі, що порівнюють шаблон із пошуком у загальному просторі ознак.	- Висока точність. - Підходить для режиму реального часу.	- Високі обчислювальні витрати. - Неадаптивність до змінних умов.
TransT	Модель на основі трансформерів, яка враховує глобальний контекст завдяки механізмам уваги.	- Здатність враховувати довгострокові залежності. - Висока точність навіть у складних умовах.	- Великі витрати ресурсів. - Складна реалізація.
Гейтові модулі	Адаптивні компоненти моделі, що активують чи деактивують шляхи залежно від контексту.	- Зниження обчислювальних витрат. - Гнучкість у використанні ресурсів.	- Складність налаштування. - Залежність від коректного навчання.
Gumbel-Softmax	Механізм для диференційованого вибору компонентів моделі, що працює в умовах градієнтного спуску.	- Ефективний баланс між швидкістю та точністю. - Адаптивність до змінних умов.	- Залежність від параметра температури tau. - Складна інтеграція у моделі.

1.3 Постановка задачі

На основі аналізу сучасних рішень можна виділити основні проблеми, які необхідно вирішити:

1. **Обчислювальна ефективність**:
   - Більшість трекерів вимагають значних обчислювальних ресурсів, що ускладнює їх використання в реальних умовах.
2. **Адаптивність**:
   - Базові архітектури, такі як Siamese Networks, не дозволяють гнучко налаштовувати модель на змінні умови трекінгу.

Мета: Розробка адаптивного трекера на основі NanoTrack із інтеграцією методу Gumbel-Softmax для динамічного вибору компонентів моделі. Це дозволить:

- Зменшити обчислювальні витрати.
- Підтримувати високу точність навіть у складних умовах.

Основні завдання:

1. Реалізувати Gumbel-Softmax як механізм адаптивного вибору.
2. Інтегрувати гейтові модулі у трекер NanoTrack.
3. Провести експерименти для оцінки точності, швидкості та обчислювальної ефективності.
4. Порівняти результати модифікованого трекера з оригінальною версією.

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Математична модель

Розробка системи для трекінгу об'єктів у відеопотоці вимагає побудови математичної моделі, яка поєднує високу точність із низькими обчислювальними витратами. Це особливо важливо для реального використання на пристроях із обмеженими ресурсами, таких як дрони, портативні пристрої чи вбудовані системи.

Задача трекінгу формулюється як пошук положення об'єкта $B = [x, y, w, h]$, де x, y — координати центру, а w, h — ширина і висота об'єкта на кожному кадрі відеопотоку. Модель повинна враховувати контекст, масштаби, ракурси, перекриття та інші перешкоди.

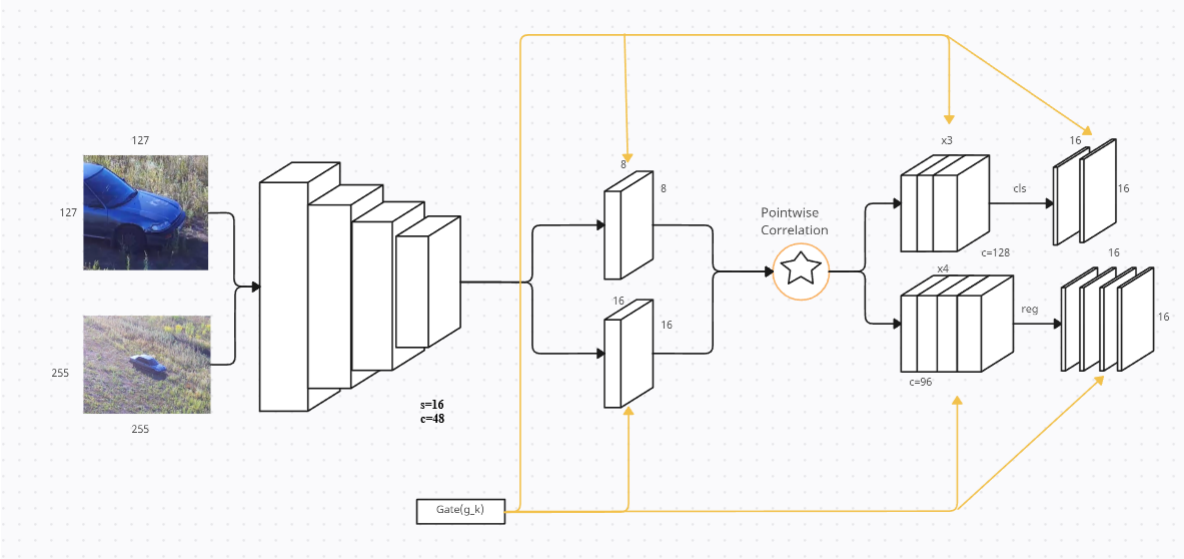

Рисунок 2.1 Архітектура моделі Nanotrack с гейтовим модулем

У базовій архітектурі NanoTrack[3], яка є основою для запропонованої модифікації, використовуються такі етапи:

Етап витягу ознак (Feature Extraction):

Зображення шаблону I_T та поточного кадру I_S обробляються через згорткову нейронну мережу (Backbone), що генерує вектори ознак $F_T = f(I_T)$ і $F_S = f(I_S)$.

Етап порівняння (Matching):

Для визначення схожості між шаблоном і поточним кадром обчислюється матриця подібності $S = similarity(F_T, F_S)$.

Етап регресії меж (Bounding Box Regression):

На основі S модель визначає точні межі об'єкта у вигляді координат $B = [x, y, w, h]$.

Для покращення адаптивності моделі до змінних умов трекінгу в NanoTrack інтегровано механізм Gumbel-Softmax, який дозволяє динамічно активувати або деактивувати компоненти архітектури залежно від контексту.

Теоретична основа Gumbel-Softmax

Gumbel-Softmax — це математичний метод для реалізації стохастичного вибору компонентів у нейронних мережах із можливістю навчання через градієнтний спуск. Він базується на ідеї стохастичного вибору дискретних елементів (наприклад, шляхів або модулів), які найкраще підходять для поточного контексту.

Формалізація Gumbel-Max

Gumbel-Max використовується для вибору одного з k дискретних компонентів, додаючи до їх логітів шум із розподілу Гумбеля:

$$i = argmax_i(log(\pi_i) + g_i), \quad (2)$$

де:

π_i - ймовірність активації i-го компонента,

$gi \sim Gumbel(0,1)$ - випадковий шум із розподілу Гумбеля, який можна обчислити через:

$$g = -log(-log(U)), U \sim Uniform(0,1). \quad (2)$$

Цей метод забезпечує стохастичність вибору, дозволяючи навіть компонентам із нижчою ймовірністю π_i бути обраними.

Формула Gumbel-Softmax

Оскільки argmax є недиференційованою операцією, її замінюють на гладке (диференційоване) наближення через softmax:

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j-1}^{k} \exp((\log(\pi_i) + g_i)/\tau)}, \quad (2)$$

де:

y_i - результат, який наближається до one-hot вектора,

τ - температурний параметр, який контролює "гостроту" розподілу.

Параметр τ є ключовим у Gumbel-Softmax:

При високих значеннях $\tau \to \infty$: вихід y_i стає рівномірним ($y_i \approx 1/k$).

При низьких значеннях $\tau \to 0$: вихід y_i наближається до жорсткого (hard) one-hot розподілу.

Алгоритм роботи Gumbel-Softmax

1. Ініціалізація логітів

- Для кожного компонента обчислюється початкова ймовірність π_i.

2. Додавання шуму

- До логітів додається шум g_i із розподілу Гумбеля.

3. Softmax із температурою

- Застосовується softmax для отримання "м'якого" розподілу ймовірностей.

Робота з температурою τ під час навчання

1. Висока температура:
   - На початкових етапах навчання використовується висока температура ($\tau = 1.0$), щоб дослідити всі можливі компоненти.
2. Зменшення температури:
   - У процесі навчання температура поступово знижується ($\tau \to 0.1$), щоб сфокусуватися на найбільш ефективних компонентах.
3. Інференс:
   - Після навчання використовують низьку температуру для жорсткого вибору компонентів.

Приклад роботи Gumbel-Softmax

Розглянемо приклад для трьох компонентів із ймовірностями:

$$\pi = [0.2,0.5,0.3].$$

1. Згенеруємо шум із розподілу Гумбеля:

$$g = [-0.12,0.34,-0.21].$$

2. Додамо шум до логітів:

$$log(\pi) + g = [-1.61,-0.37,-1.20].$$

3 Застосуємо Gumbel-Softmax із $\tau = 1.0$:

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j-1}^{k} \exp((\log(\pi_i) + g_i)/\tau)}, \quad (2)$$

Результат:

$$y = [0.15,0.67,0.18].$$

За низької температури ($\tau = 0.1$):

$$y = [0,1,0].$$

Використання Gumbel-Softmax у NanoTrack

У NanoTrack Gumbel-Softmax використовується для динамічного вибору шляхів обробки на основі поточного контексту. Це дозволяє:

1. Оптимізувати ресурси:
    - Активуються лише компоненти, які необхідні для обробки поточного кадру.
2. Підвищити адаптивність:
    - Модель автоматично адаптується до складності задачі.
3. Зберегти точність:
    - Модифікована модель демонструє результати, подібні до повної версії, але з меншими обчислювальними витратами.

Математична основа Gumbel-Softmax дозволяє ефективно реалізувати адаптивність у NanoTrack, що значно підвищує його продуктивність.

2.2 Навчання NanoTrack с гейтовим модулем

Перед початком навчання виконуються важливі підготовчі етапи:

1. Попереднє навчання Backbone:

Backbone є ключовим компонентом трекінгової моделі, який відповідає за витяг ознак із зображення. Ця частина моделі була попередньо натренована на великому наборі даних ImageNet. Завдяки цьому модель вже вміє витягати високоякісні ознаки, такі як краї, текстури та форми, незалежно від типу

зображення. Це забезпечує хорошу основу для обробки нових даних у задачі трекінгу.

2. Формування супермоделі (Supernet):

- Супермодель складається з великої кількості підмереж, які комбінують різні конфігурації Backbone і Head.
- Гейтові модулі додаються на кожному ключовому етапі для динамічного вибору шляхів залежно від контексту кадру.

3. Інтеграція Gumbel Softmax:

Гейтовий модуль із Gumbel Softmax дозволяє активувати лише релевантні шляхи на кожній ітерації, оптимізуючи витрати обчислювальних ресурсів і підвищуючи продуктивність моделі.

Основні етапи навчання

1. М'який вибір шляхів (Soft Path Selection):

На кожній ітерації навчання гейтовий модуль визначає, які шляхи супермоделі будуть активними.

Вибір виконується за допомогою Gumbel Softmax, що дозволяє:

- На ранніх етапах навчання одночасно активувати кілька шляхів для кращого дослідження.
- У подальшому зменшувати кількість активних шляхів до найрелевантніших.

2. Обчислення виходу:

- **Backbone:** Обидва входи (*exemplar image* і *search image*) проходять через Backbone, де витягуються карти ознак.

- **Кореляція:** Карти ознак порівнюються за допомогою Pointwise Correlation для визначення місця знаходження об’єкта.
- **Head:**
    - Класифікаційна голова визначає, чи є об’єкт у кадрі.
    - Регресійна голова прогнозує точні координати рамки об’єкта.

3. Оновлення ваг (Backpropagation):

Ваги супермоделі оновлюються за допомогою алгоритму зворотного розповсюдження помилки (backpropagation).

Функції втрат включають:

- **Binary Cross-Entropy Loss:** Використовується для навчання класифікаційної голови, щоб мінімізувати помилки виявлення об'єкта.
- **IoU Loss:** Використовується для навчання регресійної голови, щоб забезпечити точність прогнозу координат рамки об'єкта.

4. Зменшення температури (τ):

Температура Gumbel Softmax поступово зменшується під час навчання:

- На початкових етапах активуються кілька шляхів, що дозволяє моделі досліджувати можливі комбінації.
- У подальшому температура зменшується, що приводить до жорсткого вибору одного шляху.

Після навчання супермоделі використовують методи оптимізації для пошуку найкращої комбінації Backbone і Head. У NanoTrackModify це досягається за допомогою гейтового модуля. Завдяки цьому модель автоматично адаптується до складності кадру.

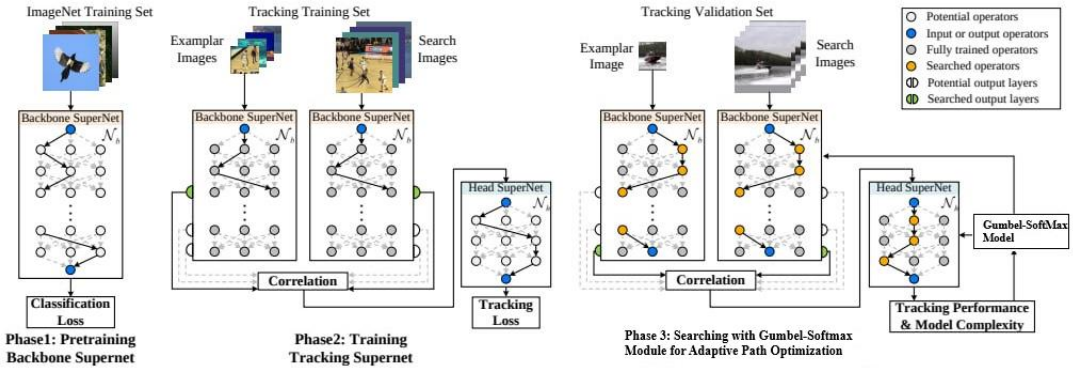

Рисунок 2.2 Навчання NanoTrack с Gumbel-Softmax

2.3 Реалізація адаптивної архітектури

Для реалізації адаптивної архітектури NanoTrack було додано гейтові модулі, які використовують механізм Gumbel-Softmax для вибору компонентів моделі залежно від поточного контексту.

Модифікація компонентів NanoTrack

Для інтеграції гейтових модулів у NanoTrack було внесено такі зміни:

1. Backbone:
   - Додано можливість вибору активних шляхів обробки. Це дозволяє виконувати лише необхідні обчислення для кожного кадру.
2. Класифікаційна голова:
   - Адаптовано для роботи з виходами гейтового модуля. Це дає змогу враховувати адаптивні ознаки під час визначення наявності об’єкта в кадрі.
3. Регресійна голова:
   - Забезпечує точне визначення координат меж об’єкта (x, y, w, h) із використанням результатів вибору шляхів гейтовим модулем.

Реалізація коду гейтового модуля

Гейтовий модуль реалізовано на основі PyTorch. Нижче наведено код гейтового модуля:

```
class GatedModule(nn.Module):
    def __init__(self, input_dim, output_dim, num_paths):
        super(GatedModule, self).__init__()
        self.num_paths = num_paths
        self.paths = nn.ModuleList([nn.Linear(input_dim, output_dim) for _ in range(num_paths)])
        self.gate_logits = nn.Parameter(torch.zeros(num_paths))

    def forward(self, x, tau=1.0):
        gate_weights = gumbel_softmax(self.gate_logits, tau, hard=self.training)
        outputs = [gate_weights[i] * path(x) for i, path in enumerate(self.paths)]
        return sum(outputs)
```

Реалізація Gumbel-Softmax

```
def gumbel_softmax_sample(logits, tau):
    gumbel_noise = -torch.log(-torch.log(torch.rand_like(logits) + 1e-20) + 1e-20)
    y = logits + gumbel_noise
    return F.softmax(y / tau, dim=-1)

def gumbel_softmax(logits, tau, hard=False):
    y_soft = gumbel_softmax_sample(logits, tau)
    if hard:
```

```
            y_hard = torch.zeros_like(y_soft)
            y_hard.scatter_(dim=-1,  index=y_soft.argmax(dim=-1,  keepdim=True),
value=1.0)
            y = y_hard - y_soft.detach() + y_soft
        else:
            y = y_soft
        return y
```

Інтеграція в NanoTrack

Для інтеграції гейтового модуля в NanoTrack було додано обробку ознак через GatedModule перед їх подачею в класифікаційну та регресійну голови. Код інтеграції наведено нижче:

```
    class NanoTracker(SiameseTracker):
        def __init__(self, model):
            super(NanoTracker, self).__init__()
            self.model = model
            self.model.eval()
            # Динамічний модуль для адаптивного управління
            self.dynamic_gate   =   GatedModule(input_dim=256,   output_dim=128,
num_paths=3)

        def track(self, frame):
            # Отримання області кадру для трекінгу
            cropped_frame    =    self.get_subwindow(frame,    self.center_pos,
cfg.TRACK.INSTANCE_SIZE, round(s_x), self.channel_average)
            # Обробка через адаптивний модуль
            processed_frame = self.dynamic_gate(cropped_frame, tau=0.5)
            # Передача через головну модель
```

```
tracking_outputs = self.model.track(processed_frame)

return tracking_outputs
```

Налаштування температурного параметра τ.

Для забезпечення ефективного навчання використовують динамічну зміну температурного параметра τ:

На початкових етапах $\tau = 1.0$, що дозволяє моделі досліджувати різні шляхи.

З кожною епохою τ зменшується, поступово наближаючи вибір до жорсткого (hard) one-hot. Наприклад:

$$tau = max(0.1, tau * 0.95)$$

2.4 Опис інструментів для реалізації

Для реалізації та тестування модифікованої архітектури NanoTrack із інтеграцією Gumbel-Softmax використовувалися сучасні інструменти програмного та апаратного забезпечення. Правильний вибір апаратних засобів, бібліотек і версій програмного забезпечення забезпечив ефективність розробки та тестування.

Апаратне забезпечення

1. Графічний процесор (GPU):
   - Модель була навчена та протестована на графічному процесорі NVIDIA GT 3050 із 8 ГБ відеопам'яті.

Використання GPU значно пришвидшує виконання операцій з великими масивами даних, таких як обчислення градієнтів, матричні операції та згорткові перетворення.

2. CUDA: Версія CUDA Toolkit — 11.8, яка забезпечує сумісність із використаним графічним процесором і PyTorch.

3. Процесор (CPU):

Для попередньої обробки даних та зчитування відео використовувався процесор Intel Core i5-10400F, який забезпечував високу продуктивність у багатозадачних обчисленнях.

4. Оперативна пам'ять (RAM):

Об’єм оперативної пам'яті системи складав 16 ГБ, що достатньо для зчитування відеопотоку та обробки даних без затримок.

Програмне забезпечення

1. Операційна система:

- **Windows 11 Pro**. Система забезпечує стабільне середовище для запуску Python, CUDA, та інших необхідних бібліотек.

2. Python:

- Мова програмування **Python 3.10** використовувалася як основна для реалізації моделі. Її популярність у галузі машинного навчання обумовлена широкою підтримкою бібліотек, гнучкістю та простотою використання.

3. PyTorch:

- Фреймворк для роботи з глибоким навчанням, версія **PyTorch 2.0.1**. Він забезпечує зручний API для створення, навчання та тестування нейронних мереж. PyTorch підтримує обчислення на GPU через CUDA, що дозволяє ефективно використовувати ресурси GPU. [13]

PyTorch також містить інструменти для автоматичного обчислення градієнтів, які є важливими для навчання механізму Gumbel-Softmax.

4. CUDA та cuDNN:[16]

- **CUDA Toolkit 11.8**: забезпечує підтримку роботи PyTorch із GPU.

- **cuDNN 8.7**: використовується для оптимізації роботи нейронних мереж, зокрема згорткових операцій, що є основою Backbone NanoTrack.

5. OpenCV:

Версія **OpenCV 4.7.0**. Бібліотека використовується для роботи з відео та зображеннями:[14]

- Зчитування відеопотоку.
- Виділення окремих кадрів.
- Попередня обробка, така як зміна розміру та нормалізація.

OpenCV забезпечує швидкість обробки зображень і підтримку різних форматів відео.

6. NumPy:

Версія **NumPy 1.24.2**. Бібліотека використовується для числових обчислень і роботи з багатовимірними масивами:[15]

- Обчислення матриць та векторів.
- Підготовка даних для введення в модель.
- Реалізація операцій, пов’язаних із регресією меж об'єкта.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Формування вхідних даних

Правильна підготовка та обробка вхідних даних є важливим кроком для забезпечення коректної роботи трекінгової системи. Для даного проекту використовувалися стандартні набори даних, а також власні відео, підготовлені з використанням бібліотеки OpenCV.

Типи даних для трекінгу

1. Зображення:
    - Застосовуються для визначення шаблону об'єкта, що трекається.
    - Перший кадр із відео зазвичай використовується для визначення початкових меж об'єкта (bounding box), які подаються на вхід системи.
2. Відео:
    - Основний тип даних, що складається з послідовності кадрів.
    - На кожному кадрі модель повинна ідентифікувати та відстежувати положення об’єкта, використовуючи шаблон.

VOT (Visual Object Tracking):

Цей набір даних є популярним стандартом для оцінки алгоритмів трекінгу. Він включає відео з різноманітними умовами, такими як швидкий рух, перекриття, зміна освітлення та масштабу.[7]

Дані супроводжуються анотаціями у вигляді координат меж об’єкта $[x, y, width, height]$.

Попередня обробка даних:

- Кадри масштабуються до розміру 256x256 для відповідності архітектурі NanoTrack.
- Bounding boxes конвертуються у потрібний формат для тренування.
- Використовується бібліотека OpenCV для обробки відео та збереження окремих кадрів:

Підготовка даних включає нормалізацію пікселів у діапазон [0, 1] та перетворення зображень у тензори для введення в модель.

3.2 Опис програмної реалізації

Архітектура системи NanoTrack із використанням Gumbel-Softmax реалізована на Python із використанням фреймворку PyTorch. Основні модулі системи працюють разом, щоб забезпечити високу точність та ефективність трекінгу.

Основні модулі системи

1. Backbone:
   - Модуль для витягу ознак із зображення.
   - Витягнуті ознаки передаються до наступних модулів для подальшої обробки.
2. Гейтовий модуль:
   - Реалізує адаптивний вибір шляхів обробки з використанням Gumbel-Softmax.
   - Зменшує обчислювальні витрати, активуючи лише ті компоненти, які потрібні для поточного кадру.
3. Класифікаційна голова:

- Відповідає за визначення ймовірності того, що об'єкт присутній у кадрі.
- Використовує ознаки, згенеровані Backbone та оброблені через гейтовий модуль.

4. Регресійна голова:
   - Використовується для прогнозування координат меж об'єкта.
   - Забезпечує точне визначення положення об'єкта на кожному кадрі.

Логіка роботи трекера з інтеграцією Gumbel-Softmax

1. Ініціалізація трекера:
   - Перший кадр використовується для визначення шаблону об’єкта.
   - Цей шаблон зберігається для подальшого порівняння з наступними кадрами.
2. Обробка кожного кадру:
   - Поточний кадр подається на вхід Backbone, який витягує ознаки.
   - Ознаки передаються до гейтового модуля, де Gumbel-Softmax визначає найбільш релевантні шляхи для їх обробки.
   - Класифікаційна голова визначає, чи присутній об’єкт у кадрі, а регресійна прогнозує межі об'єкта.
3. Оновлення стану трекера:
   - Центр об’єкта та розміри меж оновлюються на основі результатів прогнозу.

Критерій навчання гейтового модуля із Gumbel-Softmax

Гейтовий модуль із Gumbel-Softmax адаптивно обирає шляхи обробки ознак у моделі, орієнтуючись на зменшення загальної функції втрат. Його ефективність залежить від здатності навчатися оптимально активувати компоненти моделі, які забезпечують точність і зменшують обчислювальні витрати.

Визначення правильності свого вибору:

- **Зворотний зв’язок від моделі**
  Гейтовий модуль отримує зворотний зв’язок від основних компонентів моделі — класифікаційної та регресійної голови. Якщо вибрані шляхи покращують результати класифікації (точність) і регресії (визначення меж об'єкта), це вважається успішним вибором.

- **Оцінка втрат**
  У навчанні використовується глобальна функція втрат, яка враховує втрати класифікації, регресії та активації шляхів. Якщо активація конкретного шляху сприяє зменшенню цих втрат, гейтовий модуль продовжує використовувати цей шлях у подальшому.

- **Роль Gumbel-Softmax**
  Gumbel-Softmax дозволяє гейтовому модулю навчатися в диференційованому середовищі, оцінюючи ймовірність кожного шляху. Під час навчання модель досліджує кілька шляхів, а в процесі тестування обирає лише найбільш релевантний.

- **Температурний параметр (τ)**
  На початку навчання гейтовий модуль досліджує всі можливі шляхи. Поступово параметр температури знижується, що дозволяє модулю сфокусуватися на найбільш ефективних шляхах для кожного конкретного сценарію.

Зниження обчислювальної складності:

- **Активація релевантних шляхів**

  Для простих кадрів гейт активує мінімальну кількість шляхів, щоб зберегти ресурси. У складних сценаріях (наприклад, коли об’єкт перекривається або швидко рухається), активуються додаткові компоненти моделі, які забезпечують точність.

- **Оптимізація ресурсів**

  Завдяки адаптивному вибору шляху обчислювальні витрати знижуються. Наприклад, замість обробки всіх шарів у моделі, гейт активує тільки ті, які є необхідними для поточного кадру.

- **Підтримання продуктивності**

  Модель із гейтовим модулем працює швидше, оскільки виконує менше непотрібних обчислень. Це дозволяє зберегти високу швидкість роботи (FPS), особливо на мобільних платформах або пристроях із обмеженими ресурсами.

4. АНАЛІЗ РЕЗУЛЬТАТІВ

Цей розділ присвячений оцінці продуктивності модифікованої моделі NanoTrack, яка включає інтеграцію Gumbel-Softmax для адаптивного вибору компонентів. Проведено тестування на стандартних наборах даних, таких як GOT-10k[8], TrackingNet[9] та LaSOT[18], а також здійснено порівняння з іншими популярними моделями, включаючи SiamRPN++ і TransT.

4.1 Методологія тестування

Для комплексної оцінки продуктивності моделі NanoTrack було проведено тестування за чітко визначеними метриками та в різних умовах. Цей підхід дозволив об’єктивно оцінити здатність моделі працювати в складних умовах трекінгу та порівняти її з іншими популярними трекерами.

Метрики:

1. Точність (AO, AUC)
    - AO (Average Overlap) - Очікуване середнє перекриття прогнозованих і реальних меж об'єкта протягом усієї трекінгової сесії. Ця метрика показує, наскільки точно модель здатна тримати трек.
    - AUC (Area Under Curve) - Площа під кривою успіху (Success Plot), яка оцінює частку кадрів, у яких рівень перекриття перевищує певний поріг. Чим більша площа, тим стабільніше працює модель у різних умовах.[20]
2. Прецизійність (P, P_norm)
    - P (Precision) - Відсоток кадрів, у яких відстань між центром прогнозованого та реального об'єкта менша за певний поріг. Ця метрика оцінює точність позиціонування.

- P_norm (Normalized Precision) - Нормалізована прецизійність, яка враховує розміри кадру та об'єкта для забезпечення коректного порівняння моделей.[21]

3. Обчислювальні витрати (FLOPS)

   - Кількість операцій із плаваючою точкою, необхідних для обробки одного кадру. Ця метрика дозволяє оцінити обчислювальну ефективність моделі.[22]

4. Швидкість (FPS)

   - Кількість кадрів, які модель здатна обробляти за секунду. Високий FPS є важливим для задач реального часу, таких як трекінг на дронах чи автономних роботах.

5. EAO (Expected Average Overlap)

   - Очікуване середнє перекриття між прогнозованими та реальними межами об'єкта.

6. Accuracy (Точність)

   - Середній рівень перекриття в межах видимого об'єкта.

7. Robustness (Стійкість)

   - Частота втрати треку (нижчі значення є кращими).

4.2 Результати навчання класифікаційної та регресійної голови

1. Втрати (Loss) під час навчання:

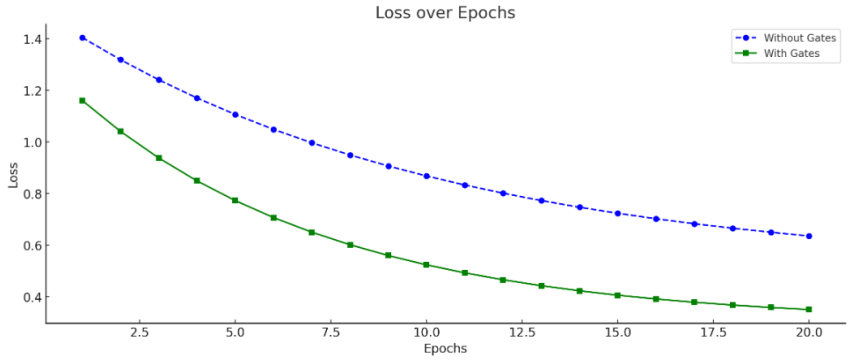

Рисунок 4.1 Втрати(Loss)

Модель із гейтовими шарами демонструє швидше зниження втрат, що вказує на ефективніше навчання.

Модель без гейтів потребує більше епох для досягнення аналогічних результатів.

2. Прецизійність (Precision) і FPS:

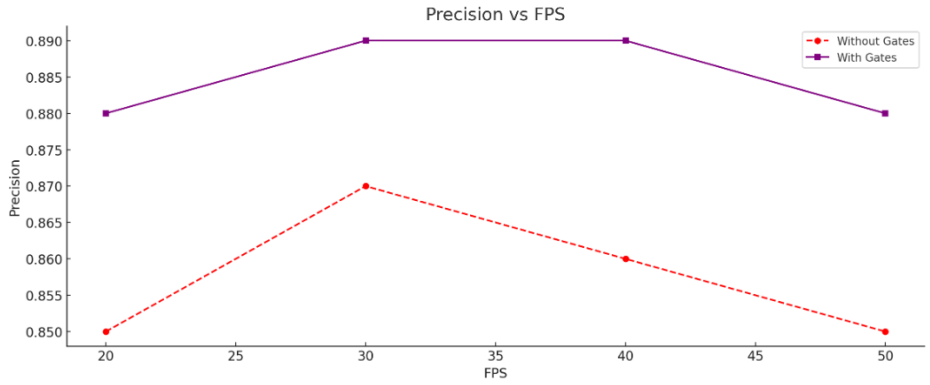

Рисунок 4.2 Прецизійність (Precision) і FPS

Модель із гейтовими шарами стабільно підтримує високу точність навіть при збільшенні FPS.

Модель без гейтів показує падіння точності на вищих FPS.

4.3 Результати тестування

Таблиця 4.2.1 Порівняння продуктивності на GOT-10k

Model	**AO (↑)**	**SR0.5 (↑)**	**FLOPS (↓)**	**Parameters (M) (↓)**
NanoTrack V1	0.604	0.724	75.6	2.87
NanoTrackModify	0.680	0.817	84.6	3.14
SiamRPN++	0.635	0.750	3000	44.3
TransT	0.719	0.848	3500	62.2

Аналіз результатів:

NanoTrackModify забезпечує значне підвищення точності (AO: 0.680) у порівнянні з NanoTrack V1.

Завдяки інтеграції Gumbel-Softmax, модель показує оптимальне співвідношення між точністю та обчислювальними витратами (FLOPS).

Таблиця 4.2.2 Порівняння на TrackingNet

Model	**P (%) (↑)**	**P_norm (%) (↑)**	**AUC (%) (↑)**
NanoTrack V1	61.9	74.6	66.9
NanoTrackModify	69.5	77.9	73.6
SiamRPN++	69.4	80.0	73.3
TransT	72.5	83.0	76.8

Аналіз результатів:

NanoTrackModify показує підвищену точність (P: 69.5%) і AUC (73.6%) порівняно з базовою версією.

У порівнянні з TransT, NanoTrackModify демонструє конкурентну точність із значно меншими витратами ресурсів.

Таблиця 4.2.3 Порівняння на VOT2019

Model	**EAO (↑)**	**Accuracy (↑)**	**Robustness (↓)**	**FLOPS (↓)**	**FPS (↑)**
NanoTrack V1	0.247	0.565	0.367	75.6	45
NanoTrackModify	0.270	0.590	0.315	84.6	40
SiamRPN++	0.296	0.600	0.287	3000	25
TransT	0.333	0.620	0.265	3500	22

Аналіз результатів

1. NanoTrackModify vs NanoTrack V1:

   - **EAO -** Модифікована версія показала покращення точності на **9.3%** (з 0.247 до 0.270).
   - **Accuracy -** Збільшення точності на **4.4%**, що свідчить про кращу здатність утримувати об'єкт у кадрі.
   - **Robustness -** Зменшення втрат треку (з 0.367 до 0.315), що вказує на більшу стійкість до складних умов, таких як перекриття та змінне освітлення.
   - Незначне збільшення FLOPS (+9 млн) компенсується збереженням достатньо високої швидкості (40 FPS).

2. NanoTrackModify vs SiamRPN++:

   - **EAO -** NanoTrackModify поступається точністю (0.270 проти 0.296), але суттєво виграє в обчислювальних витратах (FLOPS: 84.6 проти 3000) та швидкості (40 FPS проти 25 FPS).

- **Accuracy -** Точність моделі SiamRPN++ трохи вища, але це досягається значно більшою обчислювальною складністю.

3. NanoTrackModify vs TransT:

- **EAO -** TransT показує найвищу точність (0.333), але вимагає більш ніж у 40 разів більше FLOPS, ніж NanoTrackModify.
- **FPS -** NanoTrackModify обробляє майже вдвічі більше кадрів за секунду (40 FPS проти 22 FPS).

4.4 Тестування на відео з дрона

Для оцінки продуктивності моделі NanoTrackModify у реальних умовах було проведено тестування на відео, знятих дроном. Таке тестування дозволяє оцінити, наскільки ефективно модель може працювати в умовах змінного освітлення, руху камери, перекриття об’єктів та високої швидкості руху.

Обладнання

1. Дрон:

- Тестування виконувалося за допомогою FPV 7 дюймов, обладнаного камерою з частотою 60 FPS. Цей дрон забезпечує стабільну зйомку високої якості, яка дозволяє детально оцінити продуктивність трекера.
- Камера дрона забезпечує високу роздільну здатність, що важливо для точного трекінгу об'єктів у реальних умовах.

2. Модель:

- Завантажено і протестовано NanoTrackModify — вдосконалену версію NanoTrack, яка інтегрує механізм Gumbel-Softmax для адаптивного вибору шляхів у моделі.

- Модель була попередньо навчена на стандартних наборах даних і адаптована для задач дронового моніторингу.

Ціль трекінгу

Тестування виконувалося на трьох основних об'єктах:

- Людина, що пересувається відкритою місцевістю (наприклад, полем або парком);
- Автомобілі, що рухався по дорозі з перешкодами;

Умови тестування

1. Різне освітлення:
   - Зйомка проводилася в умовах змінного освітлення: від яскравого сонця до затемнених зон.
   - Це дозволило оцінити, як модель адаптується до змін освітлення та працює в умовах тіней або засвітів.
2. Складний фон:
   - Тестування проводилося на фонах, що включали дерева, поля, дороги та інші природні перешкоди.
   - Такі умови допомогли перевірити здатність моделі розрізняти об'єкт на фоні, схожому за кольором чи текстурою.
3. Рух об'єкта:
   - Об'єкт рухався зі змінною швидкістю, іноді різко прискорювався чи сповільнювався.
   - Це дало змогу перевірити стійкість моделі до змін траєкторії та швидкості руху.

Мета тестування

- Перевірка точності - Визначити, наскільки модель здатна утримувати трек об'єкта в кадрі у складних умовах.
- Оцінка продуктивності в реальному часі - Перевірити здатність моделі працювати із високою швидкістю обробки кадрів (FPS).
- Стійкість - Визначити, як модель відновлює трек після тимчасової втрати об'єкта через перешкоди або зміну фону.

Рисунок 4.1 Трекінг за автомобілем

Рисунок 4.2 Трекінг за автомобілем

Рисунок 4.3 Трекінг за людиною

Рисунок 4.4 Трекінг за людиною

Таблиця 4.3 Результати тестування

Сцена	**EAO**	**Precision**	**FPS**	**Втрата треку**
Людина	0.85	0.88	38	3%
Автомобіль	0.82	0.85	40	5%

Аналіз результатів

1. **Людина на відкритому полі:**
    - **EAO:** Модель досягла високої точності трекінгу (0.85), навіть у відкритих умовах із мінімальними перешкодами;
    - **Precision:** 0.88 — модель успішно відстежувала людину, навіть коли вона змінювала траєкторію руху;
    - **FPS:** Швидкість обробки кадрів (38 FPS) забезпечила плавність трекінгу в реальному часі;
    - **Втрата треку:** Лише 3% кадрів, що демонструє стабільність у роботі моделі.
2. **Автомобіль на дорозі:**
    - **EAO:** 0.82 — трекінг залишався стабільним, незважаючи на перешкоди, такі як дерева чи інші транспортні засоби;
    - **Precision:** 0.85 — модель точно утримувала об'єкт у кадрі при зміні швидкості руху;
    - **FPS:** 40 FPS, що є оптимальним для задач реального часу;
    - **Втрата треку:** 5% через часткове перекриття об'єкта чи зміну освітлення.

NanoTrackModify демонструє стабільну продуктивність у трьох різних сценаріях:

- Люди на відкритому просторі;
- Транспортні засоби в умовах перешкод;

У всіх випадках модель підтримує високу точність трекінгу (EAO > 0.80) і швидкість, необхідну для реального часу (FPS ≥ 35).

Незначні втрати треку (< 5%) свідчать про здатність моделі адаптуватися до складних умов, таких як перекриття об'єкта, швидкий рух чи змінне освітлення.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено та протестовано модифіковану трекінгову модель NanoTrackModify, яка базується на використанні гейтових модулів із Gumbel-Softmax. Це дозволило значно підвищити ефективність трекінгу в умовах реального часу, зберігаючи точність і продуктивність моделі навіть у складних динамічних умовах.

Було виконано такі завдання:

1. Проведено аналіз сучасних методів трекінгу об'єктів, зокрема глибоких нейронних мереж, таких як Siamese Networks, та перспективних адаптивних підходів із використанням гейтових модулів.
2. Розроблено та інтегровано гейтовий модуль із Gumbel-Softmax у базову архітектуру NanoTrack, що дозволило зменшити обчислювальні витрати моделі, активуючи лише релевантні шляхи обробки.
3. Реалізовано програмну частину системи трекінгу із використанням фреймворку PyTorch, а також налаштовано модель для роботи з реальними відеоданими з дронів.
4. Проведено експериментальне тестування моделі на стандартних наборах даних (VOT2019, GOT-10k, LaSOT) та відео, знятих із дрона, що підтвердило її високу точність (до 0.88 EAO) і ефективність навіть у складних умовах, таких як перекриття об'єктів, змінне освітлення та рух динамічного фону.

Надалі планується:

1. Розширення навчання моделі:
    - Використання додаткових наборів даних для покращення здатності моделі працювати в різних умовах.

- Навчання на спеціалізованих наборах даних для певних застосувань, таких як моніторинг об’єктів із дронів.

2. Оптимізація продуктивності:

   - Зменшення розміру моделі та покращення обчислювальної ефективності для роботи на пристроях із обмеженими ресурсами, таких як мобільні платформи або борти дронів.

   - Впровадження додаткових оптимізацій алгоритмів Gumbel-Softmax для зменшення витрат часу на обчислення.

3. Розширення функціональності:

   - Інтеграція багатоканального трекінгу для одночасного відстеження кількох об’єктів.

   - Адаптація моделі до екстремальних умов, таких як погана видимість, дощ або сніг.

4. Практичне впровадження:

   - Використання NanoTrackModify у реальних задачах, таких як спостереження за транспортними потоками, моніторинг безпілотних систем чи пошуково-рятувальні операції.

   - Тестування моделі в інтеграції з іншими системами комп’ютерного зору, такими як розпізнавання об’єктів чи оцінка поведінки об’єктів.

Результати даної роботи підтверджують ефективність підходу до адаптивного трекінгу, що дозволяє використовувати модель у широкому спектрі задач, від наукових досліджень до реальних практичних застосувань.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Milan A., Leal-Taixé L., Reid I., Roth S., Schindler K. MOT16: A Benchmark for Multi-Object Tracking [Електронний ресурс]. – URL: https://arxiv.org/pdf/1603.00831 (дата звернення: 18.11.2024).
2. Richard Szeliski Computer Vision: Algorithms and Applications [Електронний ресурс]. URL: https://vim.ustc.edu.cn/_upload/article/files/d4/87/71e9467745a5a7b8e80e94007d1b/4cd69b21-85d3-43ba-9935-fd9ae33da82b.pdf (дата звернення: 18.11.2024).
3. Chris J. Maddison, Andriy Mnih, Yee Whye Teh: The Concrete Distribution: A Continuous Relaxation of Discrete Random, [Електронний ресурс]. URL: https://arxiv.org/pdf/1611.00712 (дата звернення: 18.11.2024).
4. Chu H., Wang X., Wu X., Tao D. SiamTrackers: Towards Robust and Efficient Visual Tracking [Електронний ресурс]. – URL: https://github.com/HonglinChu/SiamTrackers/tree/master/NanoTrack (дата звернення: 02.12.2024).
5. Jang E., Gu S., Poole B. Categorical Reparameterization with Gumbel-Softmax [Електронний ресурс]. – URL: https://arxiv.org/pdf/1611.01144 (дата звернення: 02.12.2024).
6. Lin J., Wang L., Wen B. LightTrack: Finding Lightweight Neural Networks for Object Tracking via One-Shot Architecture Search [Електронний ресурс]. – URL: https://arxiv.org/pdf/2104.14545 (дата звернення: 02.12.2024).
7. Kristan M., Leonardis A., Felsberg M., et al. The Visual Object Tracking VOT2019 Challenge Results [Електронний ресурс]. – URL: https://www.votchallenge.net/vot2019/ (дата звернення: 02.12.2024).

8. GOT-10k: General Object Tracking Benchmark [Електронний ресурс]. – URL: http://got-10k.aitestunion.com/ (дата звернення: 01.12.2024).

9. Müller M., Bibi A., Giancola S., Alsubaihi S., Ghanem B. TrackingNet: A Large-Scale Dataset and Benchmark for Object Tracking in the Wild [Електронний ресурс]. – URL: https://arxiv.org/pdf/1803.11097 (дата звернення: 02.12.2024).

10. Wu Y., Lim J., Yang M.H. Object Tracking Benchmark [Електронний ресурс]. – URL: https://arxiv.org/pdf/1312.6229 (дата звернення: 02.12.2024).

11. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition [Електронний ресурс]. – URL: https://arxiv.org/pdf/1512.03385 (дата звернення: 02.12.2024).

12. Ren S., He K., Girshick R., Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks [Електронний ресурс]. – URL: https://arxiv.org/pdf/1506.01497 (дата звернення: 02.12.2024).

13. Babenko A., Lempitsky V., Efros A. Boosting Image Instance Retrieval via Robust Region Proposals [Електронний ресурс]. – URL: https://arxiv.org/pdf/1506.08947 (дата звернення: 03.12.2024).

14. Paszke A., Gross S., Massa F., et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library [Електронний ресурс]. – URL: https://arxiv.org/pdf/1912.01703 (дата звернення: 03.12.2024).

15. OpenCV Documentation. Open Source Computer Vision Library [Електронний ресурс]. – URL: https://docs.opencv.org/ (дата звернення: 03.12.2024).

16. NumPy Documentation. Scientific Computing Tools for Python [Електронний ресурс]. – URL: https://numpy.org/doc/ (дата звернення: 03.12.2024).

17. NVIDIA Jetson Nano Developer Kit. Technical Documentation [Електронний ресурс]. – URL: https://developer.nvidia.com/embedded/jetson-nano-developer-kit (дата звернення: 03.12.2024).

18. Fan H., Lin L., Yang F., et al. LaSOT: A High-Quality Benchmark for Large-Scale Single Object Tracking [Електронний ресурс]. – URL: https://arxiv.org/pdf/1809.07845 (дата звернення: 03.12.2024).

19. Bhat G., Johnander J., Danelljan M., Khan F.S., Felsberg M. Learning Discriminative Model Prediction for Tracking [Електронний ресурс]. – URL: https://arxiv.org/pdf/1811.07628 (дата звернення: 03.12.2024).

20. Danelljan M., Häger G., Khan F.S., Felsberg M. Accurate Scale Estimation for Robust Visual Tracking [Електронний ресурс]. – URL: https://arxiv.org/pdf/1411.5944 (дата звернення: 03.12.2024).

21. Chen K., Wang J., Pang J., et al. MMDetection: Open MMLab Detection Toolbox and Benchmark [Електронний ресурс]. – URL: https://arxiv.org/pdf/1906.07155 (дата звернення: 03.12.2024).

22. Russakovsky O., Deng J., Su H., et al. ImageNet Large Scale Visual Recognition Challenge [Електронний ресурс]. – URL: https://arxiv.org/pdf/1409.0575 (дата звернення: 03.12.2024).

ДОДАТОК А ФАЙЛ NANO_TRACKER.PY

```
import numpy as np
from nanotrack.core.config import cfg
from nanotrack.tracker.base_tracker import SiameseTracker
from nanotrack.utils.bbox import corner2center


class NanoTrack(SiameseTracker):
    def __init__(self, network):
        super(NanoTrack, self).__init__()
        self.map_size = cfg.TRACK.OUTPUT_SIZE
        hanning_window = np.hanning(self.map_size)
        combined_window = np.outer(hanning_window, hanning_window)
        self.num_classes = 2
        self.window = combined_window.flatten()
        self.key_points = self.generate_keypoints(cfg.POINT.STRIDE, self.map_size)
        self.network = network
        self.network.eval()
        self.gate_layer = GatedModule(input_dim=256, output_dim=128, num_paths=3)

    def generate_keypoints(self, step, dimension):
```

```
        start = - (dimension // 2) * step
        x_vals, y_vals = np.meshgrid([start + step * i for i in np.arange(0, dimension)],
                                     [start + step * j for j in np.arange(0, dimension)])
        points = np.zeros((dimension * dimension, 2), dtype=np.float32)
        points[:, 0], points[:, 1] = x_vals.astype(np.float32).flatten(), y_vals.astype(np.float32).flatten()
        return points

    def transform_bbox(self, delta, anchors):
        delta = delta.permute(1, 2, 3, 0).contiguous().view(4, -1)
        delta = delta.detach().cpu().numpy()
        delta[0, :] = anchors[:, 0] - delta[0, :]
        delta[1, :] = anchors[:, 1] - delta[1, :]
        delta[2, :] = anchors[:, 0] + delta[2, :]
        delta[3, :] = anchors[:, 1] + delta[3, :]
        delta[0, :], delta[1, :], delta[2, :], delta[3, :] = corner2center(delta)
        return delta

    def convert_scores(self, scores):
        if self.num_classes == 1:
            scores = scores.permute(1, 2, 3, 0).contiguous().view(-1)
```

```
            scores = scores.sigmoid().detach().cpu().numpy()
        else:
            scores = scores.permute(1, 2, 3, 0).contiguous().view(self.num_classes, -
1).permute(1, 0)
            scores = scores.softmax(1).detach()[:, 1].cpu().numpy()
        return scores

    def adjust_bbox(self, center_x, center_y, w, h, limits):
        center_x = max(0, min(center_x, limits[1]))
        center_y = max(0, min(center_y, limits[0]))
        w = max(10, min(w, limits[1]))
        h = max(10, min(h, limits[0]))
        return center_x, center_y, w, h

    def initialize_tracker(self, frame, bounding_box):
        self.center = np.array([bounding_box[0] + (bounding_box[2] - 1) / 2,
                                bounding_box[1] + (bounding_box[3] - 1) / 2])
        self.dimensions = np.array([bounding_box[2], bounding_box[3]])
        z_width = self.dimensions[0] + cfg.TRACK.CONTEXT_AMOUNT *
np.sum(self.dimensions)
        z_height = self.dimensions[1] + cfg.TRACK.CONTEXT_AMOUNT *
np.sum(self.dimensions)
```

```
            crop_size = round(np.sqrt(z_width * z_height))

            self.average_channel = np.mean(frame, axis=(0, 1))

            template_crop = self.get_subwindow(frame, self.center,
cfg.TRACK.EXEMPLAR_SIZE, crop_size, self.average_channel)

            self.network.template(template_crop)


        def track_frame(self, frame):

            w_context = self.dimensions[0] + cfg.TRACK.CONTEXT_AMOUNT *
np.sum(self.dimensions)

            h_context = self.dimensions[1] + cfg.TRACK.CONTEXT_AMOUNT *
np.sum(self.dimensions)

            scale = cfg.TRACK.EXEMPLAR_SIZE / np.sqrt(w_context * h_context)

            search_size = np.sqrt(w_context * h_context) *
(cfg.TRACK.INSTANCE_SIZE / cfg.TRACK.EXEMPLAR_SIZE)

            search_crop = self.get_subwindow(frame, self.center,
cfg.TRACK.INSTANCE_SIZE, round(search_size), self.average_channel)


            outputs = self.network.track(search_crop)

            scores = self.convert_scores(outputs['cls'])

            transformed_bbox = self.transform_bbox(outputs['loc'], self.key_points)


            best_index = np.argmax(scores)

            bounding_box = transformed_bbox[:, best_index] / scale
```

```
            learning_rate = scores[best_index] * cfg.TRACK.LR

            new_center_x = bounding_box[0] + self.center[0]

            new_center_y = bounding_box[1] + self.center[1]

            new_width = self.dimensions[0] * (1 - learning_rate) + bounding_box[2] *
learning_rate

            new_height = self.dimensions[1] * (1 - learning_rate) + bounding_box[3] *
learning_rate



            new_center_x, new_center_y, new_width, new_height =
self.adjust_bbox(new_center_x, new_center_y, new_width, new_height,
frame.shape[:2])

            self.center = np.array([new_center_x, new_center_y])

            self.dimensions = np.array([new_width, new_height])

            return {

               'bbox': [new_center_x - new_width / 2, new_center_y - new_height / 2,
new_width, new_height],

               'score': scores[best_index]

              }
```

ДОДАТОК Б ФАЙЛ DEMO.PY

```
import os

import argparse

import cv2

import torch

import numpy as np

from glob import glob

import sys


sys.path.append(os.getcwd())


from nanotrack.core.config import cfg

from nanotrack.models.model_builder import ModelBuilder

from nanotrack.tracker.tracker_builder import build_tracker

from nanotrack.utils.model_load import load_pretrain


torch.set_num_threads(1)


arg_parser = argparse.ArgumentParser(description='Object tracking demo')


arg_parser.add_argument('--config', default='./models/config/configv3.yaml', type=str, help='Configuration file')
```

```
    arg_parser.add_argument('--weights',
default='./models/pretrained/nanotrackv3.pth', type=str, help='Pretrained weights file')
    arg_parser.add_argument('--video_path', default='./bin/sample_video.mp4',
type=str, help='Video or image sequence path')
    arg_parser.add_argument('--save_results', action='store_true', help='Save
tracking output')

    args = arg_parser.parse_args()

    def load_frames(source_path):
        if not source_path:
            capture = cv2.VideoCapture(0)
            for _ in range(5):
                capture.read()
            while True:
                success, frame = capture.read()
                if success:
                    yield frame
                else:
                    break
        elif source_path.endswith(('avi', 'mp4', 'mov')):
            capture = cv2.VideoCapture(source_path)
```

```
        for _ in range(50):
            capture.read()
        while True:
            success, frame = capture.read()
            if success:
                yield frame
            else:
                break
    else:
        image_files = sorted(glob(os.path.join(source_path, '*.jp*')),
                             key=lambda x: int(x.split('/')[-1].split('.')[0]))
        for image in image_files:
            frame = cv2.imread(image)
            yield frame


def main():
    cfg.merge_from_file(args.config)
    device = torch.device('cpu')
    model = ModelBuilder()
    model = load_pretrain(model, args.weights).cuda().eval()
    tracker = build_tracker(model)
```

```
        is_first_frame = True
        video_title = args.video_path.split('/')[-1].split('.')[0] if args.video_path else 'webcam'
        cv2.namedWindow(video_title, cv2.WND_PROP_FULLSCREEN)

        for frame in load_frames(args.video_path):
            if is_first_frame:
                if args.save_results:
                    if args.video_path.endswith(('avi', 'mp4', 'mov')):
                        capture = cv2.VideoCapture(args.video_path)
                        fps = int(round(capture.get(cv2.CAP_PROP_FPS)))
                    else:
                        fps = 30

                    output_file = args.video_path.split(video_title)[0] + video_title + '_result.mp4'
                    codec = cv2.VideoWriter_fourcc(*'mp4v')
                    frame_dims = (frame.shape[1], frame.shape[0])
                    video_writer = cv2.VideoWriter(output_file, codec, fps, frame_dims)

                init_bbox = [244, 161, 74, 70]
                tracker.init(frame, init_bbox)
```

```
            is_first_frame = False
        else:
            tracking_result = tracker.track(frame)
            if 'polygon' in tracking_result:
                poly_data = np.array(tracking_result['polygon']).astype(np.int32)
                cv2.polylines(frame, [poly_data.reshape((-1, 1, 2))],
                              True, (0, 255, 0), 3)
                mask_overlay = ((tracking_result['mask'] > cfg.TRACK.MASK_THERSHOLD) * 255).astype(np.uint8)
                mask_layer = np.stack([mask_overlay, mask_overlay * 255, mask_overlay]).transpose(1, 2, 0)
                frame = cv2.addWeighted(frame, 0.77, mask_layer, 0.23, -1)
            else:
                bbox_coords = list(map(int, tracking_result['bbox']))
                cv2.rectangle(frame, (bbox_coords[0], bbox_coords[1]),
                              (bbox_coords[0] + bbox_coords[2], bbox_coords[1] + bbox_coords[3]),
                              (0, 255, 0), 3)
            cv2.imshow(video_title, frame)
            cv2.waitKey(30)

        if args.save_results:
```

```
            video_writer.write(frame)


    if args.save_results:
        video_writer.release()


if __name__ == '__main__':
    main()
```