

3. РОЗГОРТАННЯ, НАВЧАННЯ ТА ОПТИМІЗАЦІЯ КОРИСТУВАЦЬКИХ МОДЕЛЕЙ

3.1. Налаштування Jetson Nano

Підготовка пристрою Jetson Nano до роботи включає кілька ключових етапів: завантаження базової операційної системи, налаштування бібліотек і програмного забезпечення, а також оптимізація системи для роботи з нейронними мережами та іншими задачами.

Першим етапом налаштування є підготовка SD-карти з операційною системою. Для цього використовується офіційний образ ОС, який записується на SD-карту за допомогою спеціальних програм. Після цього SD-карта вставляється в пристрій, і Jetson Nano підключається до джерела живлення, монітора, клавіатури та миші. Під час першого запуску користувач налаштовує мову, часовий пояс та створює обліковий запис.

Наступним кроком є оновлення операційної системи. Це дозволяє встановити останні оновлення, що забезпечують коректну роботу пристрою. Додатково перевіряється встановлення всіх компонентів JetPack SDK — програмного пакету, який включає необхідні драйвери та бібліотеки для роботи з апаратним прискоренням обчислень, такими як TensorRT, CUDA та cuDNN.

Для розробки систем із машинного навчання на Jetson Nano встановлюються бібліотеки Python, такі як NumPy, OpenCV, TensorFlow та PyTorch. Це забезпечує сумісність з основними інструментами для роботи з даними та нейронними мережами. Додатково встановлюються необхідні бібліотеки для роботи з камерою, наприклад, для обробки відео та виконання реального часу аналізу зображень.

Для роботи з моделями глибокого навчання пристрій оптимізується через активацію режиму максимальної продуктивності. Це забезпечує найкращу ефективність при виконанні ресурсоємних задач, таких як розпізнавання об'єктів чи відеоаналітика.

Завершальним етапом є тестування налаштувань. Це включає перевірку роботи камер, моделей глибокого навчання, таких як YOLO, та можливості обробки вхідних даних. Після цього створюється резервна копія конфігурації для швидкого відновлення в разі необхідності.

Таким чином, налаштування Jetson Nano — це комплексний процес, який включає встановлення та конфігурацію апаратного та програмного забезпечення, а також адаптацію системи для виконання складних задач. Це дозволяє ефективно використовувати пристрій у проектах з розпізнавання об'єктів, обробки зображень та інших завдань штучного інтелекту.

3.2. Навчання користувацьких моделей

Для створення користувацького набору даних для нейронної мережі YOLOv8 за допомогою ресурсу Roboflow, орієнтованого на розпізнавання автомобілів, виконуються наступні кроки. Спочатку створюється новий проект з відповідною назвою, наприклад, "Car Detection Dataset", обирається тип проекту "Object Detection", що підходить для розпізнавання об'єктів. Після цього здійснюється збір зображень автомобілів, які будуть використані для навчання, забезпечується різноманітність даних з різних ракурсів, умов освітлення та місць.

Після збору зображень, їх завантажують до створеного проекту в Roboflow.

Завантаження можна здійснювати масово для скорочення часу. Наступним

кроком є анотація даних безпосередньо на платформі Roboflow (

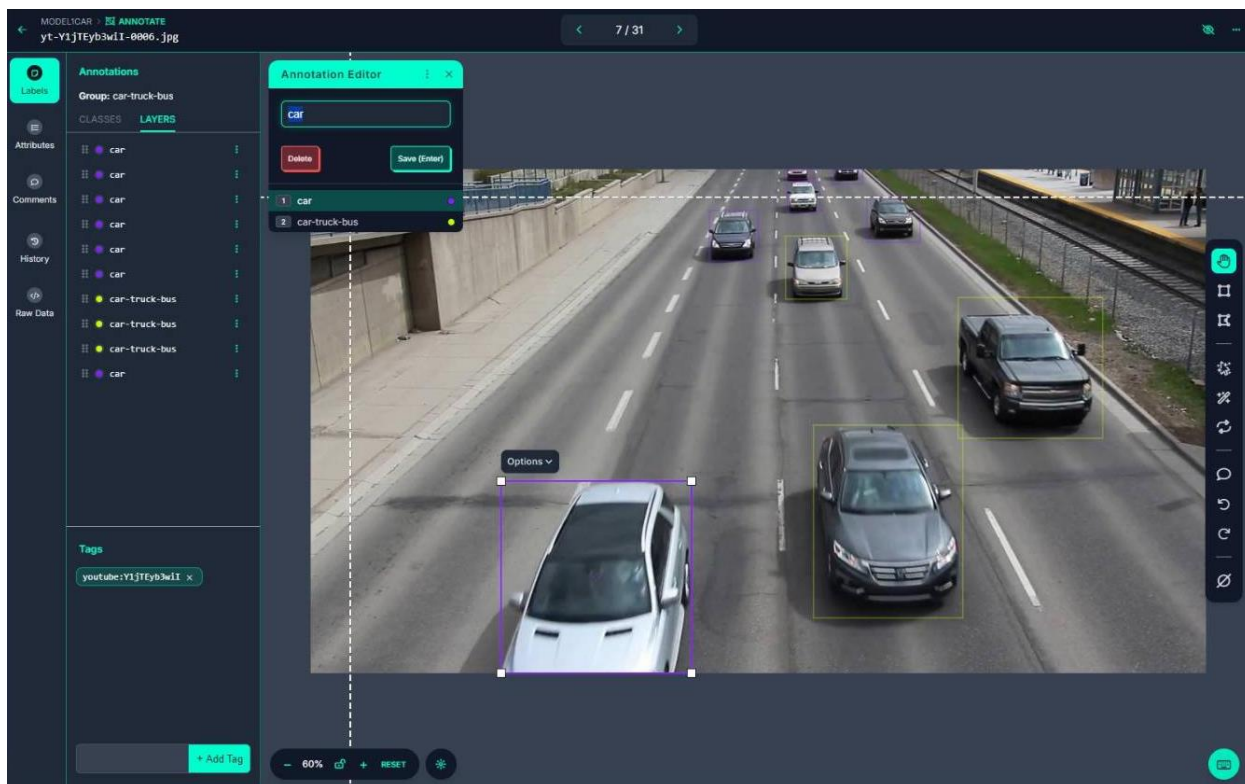


Рис. 3.1), де для кожного зображення відзначаються об'єкти (автомобілі) у вигляді прямокутних рамок (bounding boxes) з відповідним тегом, наприклад, "car". Можливе використання автоматизованої анотації для прискорення процесу.

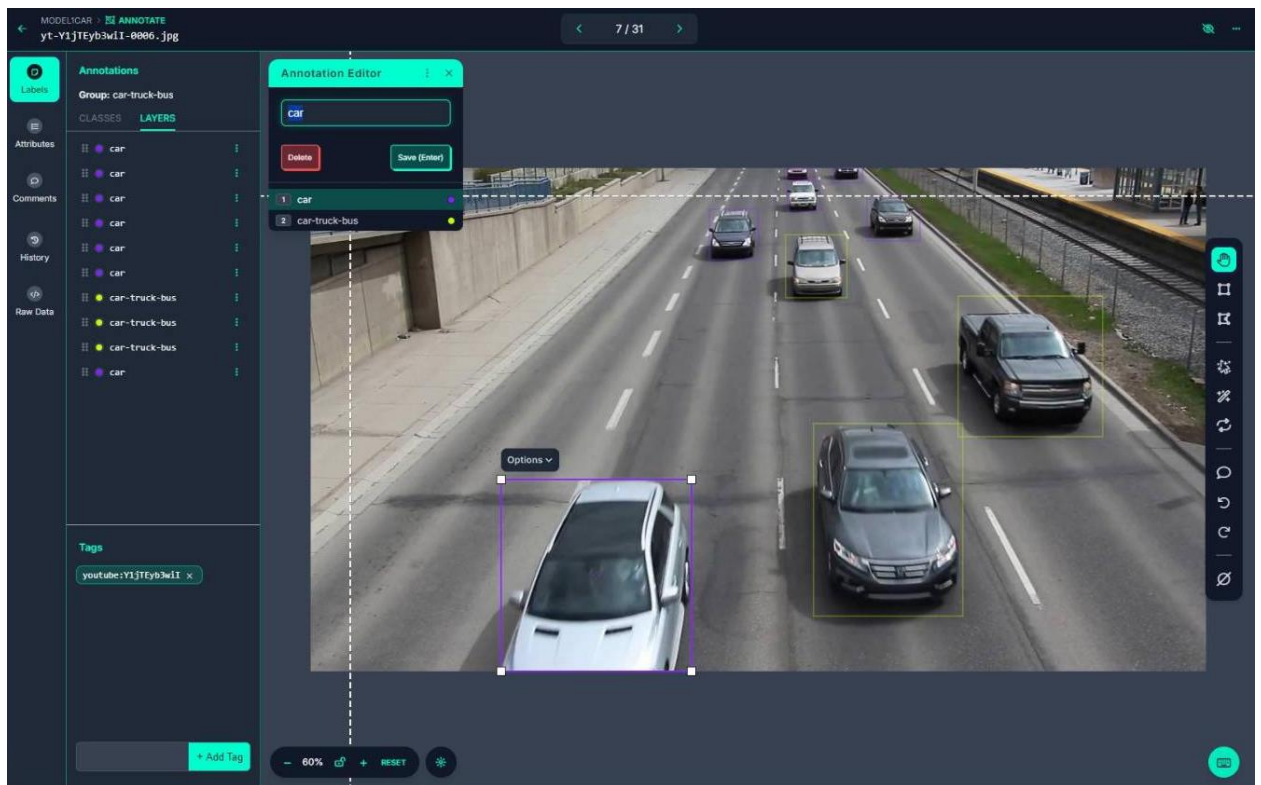


Рис. 3.1. Анотація даних в Roboflow

Обмежувальні рамки (bounding boxes) є ключовим елементом у задачах розпізнавання об'єктів, зокрема при навчанні моделей, таких як кастомна YOLOv8. Вони дозволяють моделі навчитися визначати, де саме на зображенні знаходяться об'єкти, завдяки чіткій локалізації. Рамки задають координати прямокутника, який обмежує об'єкт, включаючи координати центру або верхнього лівого кута, а також ширину та висоту. Крім того, кожна рамка асоціюється з міткою класу, що визначає тип об'єкта (наприклад, "машина", "людина"). Це допомагає моделі не лише локалізувати об'єкт, але й класифікувати його.

Рамки є основою для створення навчального датасету. Кожне зображення в наборі даних супроводжується анотаціями, що містять рамки та класи об'єктів. Модель використовує ці дані для регресії координат рамок, класифікації об'єктів у рамках і визначення ймовірності наявності об'єкта в певній області. У моделі YOLOv8 функція втрат базується на трьох компонентах: локалізації (точність визначення координат рамок), класифікації (вірність класу об'єкта) та об'єктності (ймовірність наявності об'єкта у передбаченій рамці).

Обмежувальні рамки допомагають моделі зосереджувати увагу на конкретних ділянках зображення, що особливо важливо для розпізнавання дрібних об'єктів. Це забезпечує покращення якості навчання та дозволяє моделі враховувати деталі навіть за умов змін освітлення, перспективи або шуму в зображеннях. Для кастомних моделей YOLOv8 це особливо корисно, адже вона здатна оптимізовано працювати з об'єктами різного розміру.

Щоб навчання моделі було ефективним, важливо створювати обмежувальні рамки з максимальною точністю, щоб вони повністю охоплювали об'єкти без зайвих просторів. Також слід забезпечити рівномірний розподіл класів у навчальному наборі даних, аби уникнути переваги одного класу над іншими, та використовувати різноманітні зображення з різними ракурсами, освітленням і масштабами. Для створення анотацій можна скористатися такими інструментами, як Roboflow.

Аугментація зображень у Roboflow — це процес створення нових зображень на основі існуючих у вашому наборі даних шляхом застосування різних трансформацій. Цей підхід допомагає покращити здатність моделі узагальнювати інформацію та ефективніше працювати з новими, раніше не баченими зображеннями. Roboflow підтримує широкий спектр аугментацій, які можна застосовувати як до всього зображення, так і до окремих об'єктів на ньому. До таких аугментацій належать дзеркальне відображення (горизонтальне або вертикальне), повороти зображення на 90 або 180 градусів, обертання на випадковий кут, обрізання випадкових частин зображення, деформації, розмиття, зміна яскравості, додавання шуму або видалення окремих областей зображення.

Щоб скористатися аугментацією, необхідно спочатку створити проект у Roboflow та завантажити вихідні дані. У процесі роботи над проектом ви можете перейти до вкладки "Versions", де є можливість додати етап аугментації. Вибираючи потрібні типи трансформацій, можна налаштувати їх параметри за допомогою повзунків. Після цього варто вказати множник аугментації, який визначає кількість нових зображень, що створюватимуться на базі одного

оригінального. Наприклад, множник 2x означає, що для кожного вихідного зображення буде створено два нових з аугментаціями. Після завершення налаштувань система автоматично згенерує нову версію набору даних.

Однією з переваг застосування аугментацій під час створення набору даних, а не під час тренування моделі, є покращена відтворюваність. Ви маєте точний запис того, які аугментації були застосовані до кожного зображення, а також зменшуєте час та ресурсні витрати під час тренування моделі. Згенерований аугментований набір даних можна використовувати для тренування моделі безпосередньо в Roboflow або експортувати для роботи в інших середовищах.

Після завершення анотації та аугментації набір даних експортується у форматі, сумісному з YOLOv8. Вибирається відповідний формат та завантажується архів із зображеннями та файлом анотацій (Рис. 3.2).

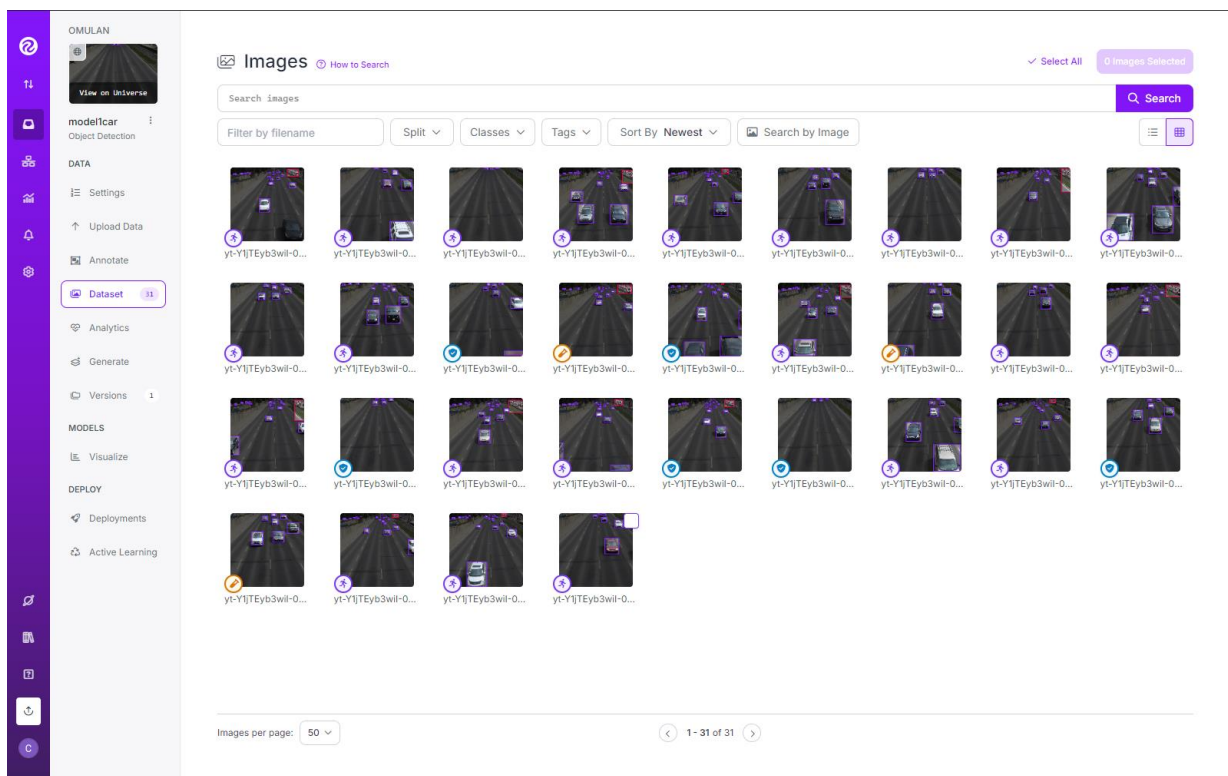


Рис. 3.2. Сформований набір даних в Roboflow

Після розпакування архіву перевіряється структура файлів на відповідність вимогам YOLOv8, яка зазвичай включає під папки "train" та "valid" з відповідними

зображеннями та анотаціями. Цей процес забезпечує створення якісного набору даних, придатного для тренування YOLOv8 на розпізнавання автомобілів.

Далі завантажується базова модель YOLO, наприклад, YOLOv8, з попередньо навченими вагами на великому загальному наборі даних, зазвичай COCO. Цей набір даних включає багато категорій об'єктів, таких як люди, тварини, транспорт тощо. Початкові ваги слугують основою для нейронної мережі, що допомагає їй розпізнавати загальні об'єкти.

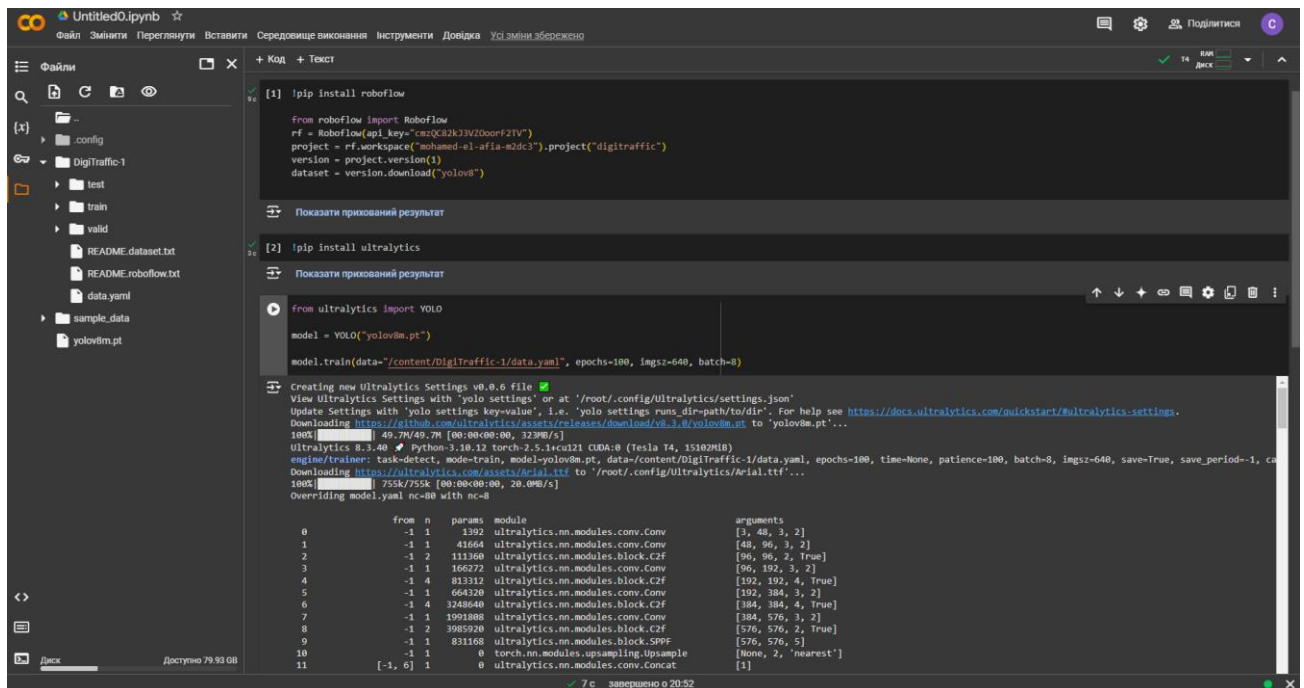
Навчання кастомної моделі YOLOv8 із готовим датасетом, створеним у Roboflow, і запуск процесу тренування за допомогою Google Colab є ефективним підходом для вирішення задач комп'ютерного зору. Цей процес включає кілька ключових етапів: підготовку даних, налаштування середовища, тренування моделі та оцінку результатів.

На першому етапі важливо підготувати датасет у Roboflow. Після завантаження зображень та відповідних анотацій проводиться обробка даних, включаючи аугментації, які підвищують стійкість моделі до різних умов. Потім датасет експортується у форматі, сумісному з YOLOv8, разом із відповідним конфігураційним файлом (наприклад, data.yaml), де вказані шляхи до тренувальних та валідаційних даних, а також список класів.

Налаштування середовища для тренування моделі виконується у Google Colab, що забезпечує доступ до ресурсів GPU для прискорення обчислень. У середовищі Colab встановлюються всі необхідні бібліотеки, такі як YOLOv8 і клієнт для роботи з Roboflow, а також завантажується датасет. Це дозволяє швидко розпочати процес навчання без потреби в локальних обчислювальних ресурсах.

Тренування моделі передбачає використання початкової архітектури YOLOv8, такої як yolov8n, яка підходить для задач з обмеженими ресурсами, або більш потужних варіантів, таких як yolov8s, залежно від вимог проєкту. Під час тренування моделі налаштовуються параметри, такі як кількість епох, розмір зображень та інші гіперпараметри рис. 3.3. Модель автоматично оцінюється на

основі метрик, таких як середня точність (mAP), що дозволяє відстежувати її ефективність.



```
[1] !pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="cszQ3zk3V2DooF2TV")
project = rf.workspace("robomart-d-e1-af1a-m2dc3").project("digitraffic")
version = project.version(1)
dataset = version.download("yolov8")

[2] !pip install ultralytics

from ultralytics import YOLO
model = YOLO("yolov8m.pt")
model.train(data="/content/DigiTraffic-1/data.yaml", epochs=100, imgsz=640, batch=8)

Creating new Ultralytics Settings v8.0.6 file ✓
View Ultralytics Settings with 'yolo settings' or at '/root/.config/Ultralytics/settings.json'
Update Settings with 'yolo settings key=value', i.e. 'yolo settings runs_dir=~/dir'. For help see https://docs.ultralytics.com/quickstart/#ultralytics-settings.
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolov8m.pt to 'yolov8m.pt'...
100% 46.7M/46.7M [00:00:00<00, 32.9MB/s]
Ultralytics 8.3.40 Python-3.10.12 torch-2.5.1rcu21 CUDA:0 (Tesla T4, 15102MiB)
engine/trainer: task=detect, mode=train, model=yolov8m.pt, data=/content/DigiTraffic-1/data.yaml, epochs=100, time=None, patience=100, batch=8, imgsz=640, save=True, save_period=-1, ca
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf'...
100% 755K/755K [00:00:00<00, 20.0MB/s]
Overriding model.yaml nc=80 with nc=8

from n params module arguments
0 -1 1 1392 ultralytics.nn.modules.conv.Conv [3, 48, 3, 2]
1 -1 1 41664 ultralytics.nn.modules.conv.Conv [48, 96, 3, 2]
2 -1 2 111360 ultralytics.nn.modules.block.C2f [96, 96, 2, True]
3 -1 1 166272 ultralytics.nn.modules.conv.Conv [96, 192, 3, 2]
4 -1 4 811312 ultralytics.nn.modules.block.C2f [192, 192, 4, True]
5 -1 1 664320 ultralytics.nn.modules.conv.Conv [192, 384, 3, 2]
6 -1 4 3248640 ultralytics.nn.modules.block.C2f [384, 384, 4, True]
7 -1 1 1991808 ultralytics.nn.modules.conv.Conv [384, 576, 3, 2]
8 -1 2 3985920 ultralytics.nn.modules.block.C2f [576, 576, 2, True]
9 -1 1 831680 ultralytics.nn.modules.block.SPPF [576, 576, 5]
10 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
11 [-1, 6] 1 0 ultralytics.nn.modules.conv.Concat [1]
```

Рис. 3.3. Тренування користувацької моделі

Після завершення тренування найкраща версія моделі зберігається і використовується для перевірки на тестових зображеннях або відео. Для цього модель виконує прогнозування на нових даних, результати якого оцінюються візуально або за допомогою метрик. Це дозволяє оцінити точність та загальну якість роботи алгоритму в умовах, близьких до реальних.

Останнім етапом є збереження та експорт навченої моделі у потрібному форматі. Для використання на пристроях з обмеженими обчислювальними можливостями, таких як Jetson Nano, модель експортується у формат TensorRT, що дозволяє оптимізувати її для швидшої роботи. Важливим аспектом є вибір компактних архітектур YOLOv8, які забезпечують ефективну продуктивність із мінімальними витратами ресурсів.

Таким чином, використання Roboflow для створення датасетів і Google Colab для тренування моделі YOLOv8 дозволяє швидко і якісно виконати задачі з

навчання моделей комп'ютерного зору. Цей підхід забезпечує гнучкість у роботі з даними, ефективність тренування та адаптацію моделі до різноманітних практичних застосувань.

Під час навчання на користувацького набору даних відбувається процес донавчання (fine-tuning). Це означає, що початкові ваги моделі адаптуються до нового набору даних, який включає лише ті класи, що є у користувацькому наборі даних, наприклад, тільки автомобілі. У процесі навчання модель оптимізує свої ваги, щоб покращити розпізнавання об'єктів, наявних у наборі даних.

Фактично базовий набір даних (наприклад, COCO) не використовується безпосередньо під час навчання на користувацькому наборі даних, проте модель використовує попередньо навчені ваги, щоб прискорити адаптацію до нового набору даних. Після донавчання на користувацького набору даних модель стає спеціалізованою на розпізнаванні саме тих об'єктів, що визначені у наборі даних. Водночас вона може зберігати деяку інформацію про класи з початкового навчання, що може бути корисним для загального розпізнавання. Таким чином, базовий набір даних використовується лише для попереднього навчання моделі, а під час навчання на користувацького набору даних відбувається адаптація моделі до специфічного набору об'єктів.

Навчання і оптимізація моделі для TensorRT є важливим етапом впровадження високопродуктивних моделей машинного навчання на пристроях з обмеженими обчислювальними ресурсами, таких як GPU NVIDIA. TensorRT дозволяє виконувати оптимізацію нейронних мереж і створювати виконувані двійки, які забезпечують швидкість і ефективність роботи. Процес створення моделі TensorRT на основі YOLOv8 складається з кількох ключових етапів.

Перший етап включає встановлення необхідного програмного забезпечення. Для роботи з TensorRT потрібно встановити бібліотеки, такі як tensorrt, onnx та додаткові модулі для оптимізації моделі. Це забезпечує підтримку формату ONNX, який використовується для передачі моделей між різними фреймворками.

Наступним кроком є експорт попередньо натренованої моделі YOLOv8 у формат ONNX. Цей відкритий стандарт дозволяє обмінюватися моделями між фреймворками, такими як PyTorch і TensorFlow. Під час експорту створюється ONNX-файл, який містить архітектуру та ваги моделі.

Після експорту модель у форматі ONNX можна оптимізувати за допомогою інструментів, таких як `onnx-simplifier` або `onnx-slim`. Ці інструменти видаляють зайві операції та підвищують продуктивність моделі. Оптимізована модель стає легшою і швидшою для подальшої обробки.

Далі модель у форматі ONNX конвертується у TensorRT. Цей процес включає створення виконуваного движка, який відповідає архітектурі апаратного забезпечення. TensorRT аналізує модель і застосовує різні оптимізації, такі як злиття шарів, зменшення точності до FP16, що дозволяє досягти значного прискорення виконання. Параметри, такі як розмір зображення та обчислювальні ресурси, налаштовуються для оптимальної продуктивності.

Після створення TensorRT-моделі проводиться її тестування. Перевірка включає оцінку точності та швидкості виконання на тестових даних. Результати порівнюються з результатами оригінальної моделі для перевірки збереження якості розпізнавання.

Створена модель TensorRT використовується на різних пристроях, таких як сервери або вбудовані системи NVIDIA Jetson. Завдяки спеціальній оптимізації модель забезпечує значно швидше виконання порівняно з оригінальною моделлю у фреймворках PyTorch або TensorFlow.

3.3. Результат навчань користувацької моделі

Нормалізована матриця плутанини, представлена на графіку рис. 3.4., використовується для оцінки якості класифікації або детекції об'єктів моделі. Вона показує, наскільки добре модель передбачає кожен клас, а також демонструє помилки у класифікації. Стовпці матриці відповідають реальним класам об'єктів, а

рядки — класам, які модель передбачила. Кольорова шкала відображає частоту або ймовірність передбачення для кожного класу, нормалізовану від 0 до 1, де 1 означає повну точність. Значення на діагоналі показують правильні передбачення моделі, тоді як позадіагональні значення вказують на помилки, коли модель класифікувала об'єкт як інший клас.

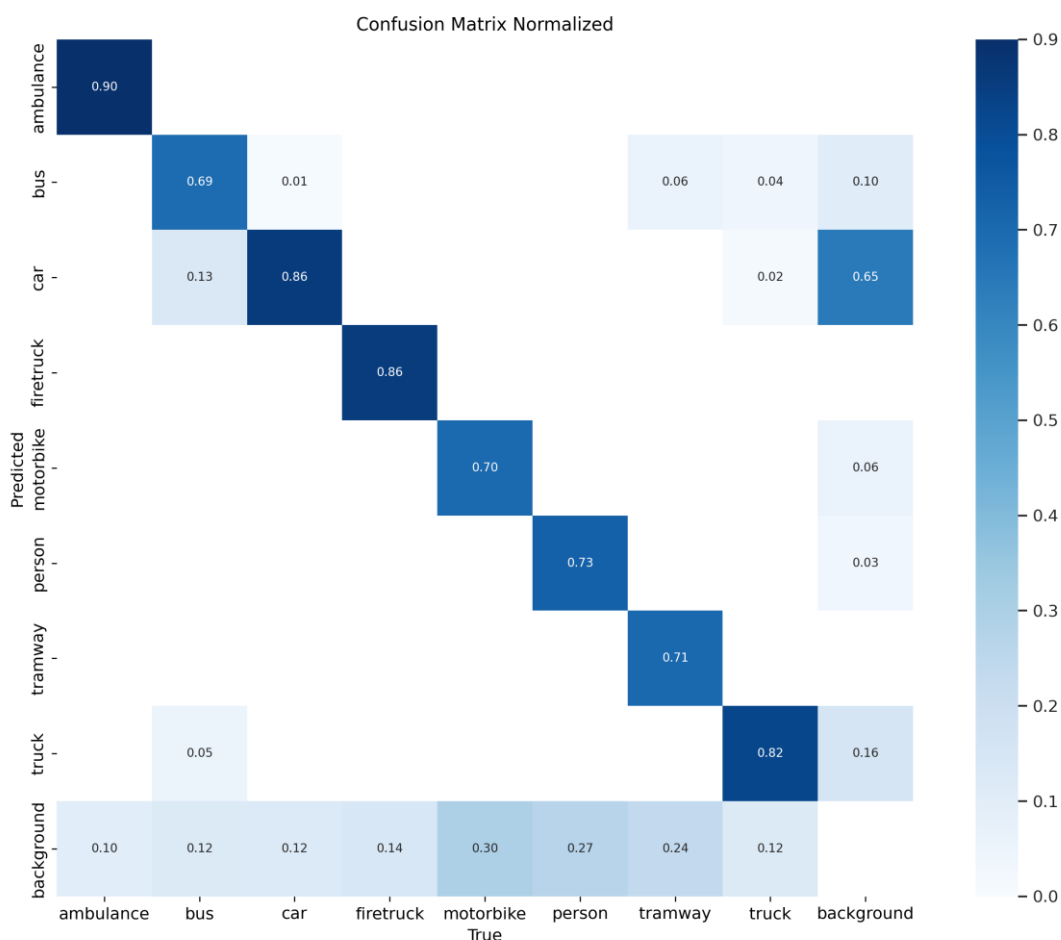


Рис. 3.4. Нормалізована матриця плутанини

Аналіз матриці плутанини свідчить, що модель демонструє високу точність для деяких класів, наприклад, "ambulance" (90%) та "firetruck" (86%), що означає, що ці об'єкти розпізнаються дуже добре. Проте для інших класів, таких як "motorbike" і "tramway", спостерігаються значні помилки. Наприклад, клас "motorbike" у 30% випадків помилково класифікується як "background", що свідчить про складнощі у його розпізнаванні в певних ситуаціях. Також

спостерігається перехресна плутанина між схожими класами, як-от "car" і "bus", де 13% зображень класу "car" помилково класифікуються як "bus". Це може бути спричинено схожістю форм або кольорів цих об'єктів у тренувальному датасеті.

Окрім того, частина класів, таких як "person", "tramway" та "truck", демонструють помилки, коли вони класифікуються як "background". Це може свідчити про недостатню деталізацію в анотаціях або брак різноманітності у тренувальному наборі даних. Високий рівень точності для класів, таких як "ambulance" і "firetruck", вказує на добре навчання моделі для цих категорій, але помилки з іншими класами свідчать про потенціал для покращення.

Графік F1-Confidence Curve демонструє залежність метрики F1 від порогу впевненості (confidence threshold) для кожного класу моделі рис. 3.5. На осі X відображено поріг впевненості, який змінюється від 0 до 1. Він показує, з якою впевненістю модель вважає передбачення правильними. На осі Y представлено значення F1-score — гармонійного середнього між точністю (precision) та повнотою (recall). Високі значення F1 вказують на хороший баланс між цими двома характеристиками.

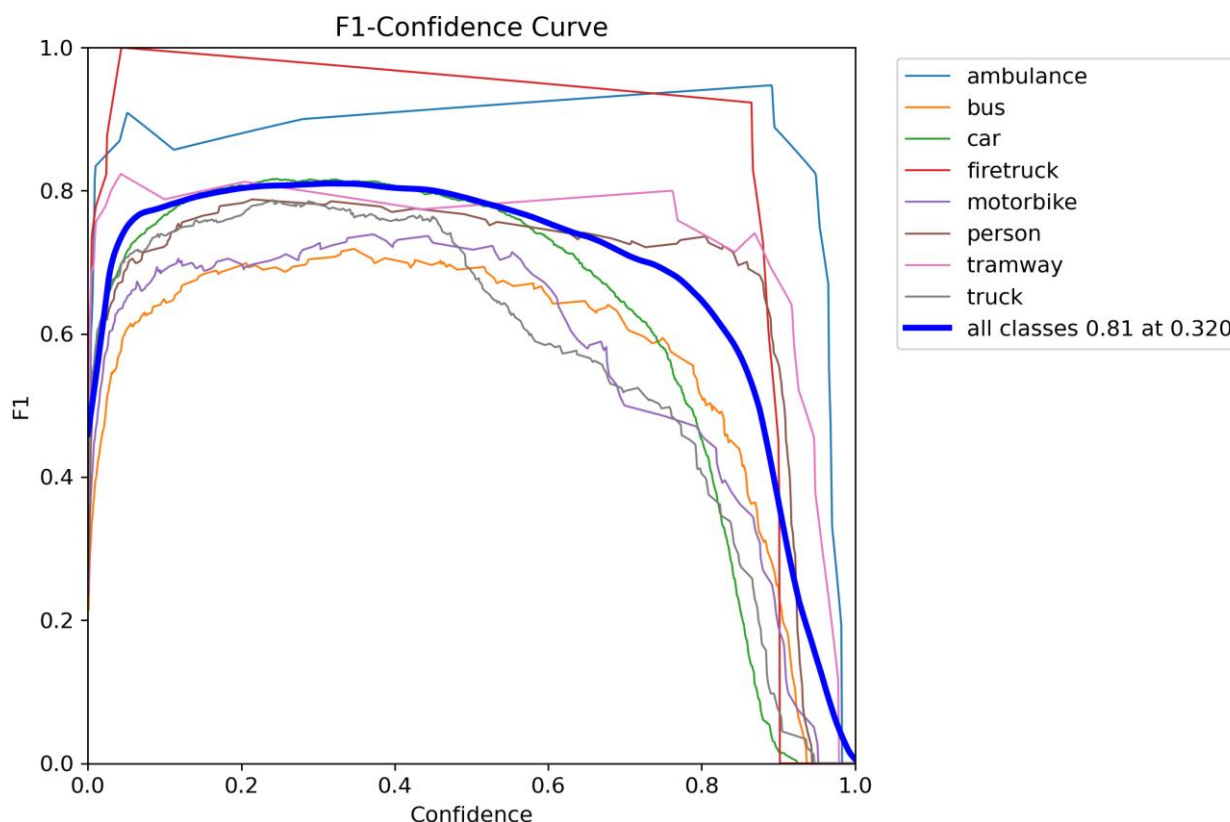


Рис. 3.5. Залежність метрики F1 від порогу впевненості

Кожна лінія на графіку представляє поведінку моделі для окремого класу. Наприклад, класи "ambulance" і "firetruck" демонструють стабільні криві з високими значеннями F1, що свідчить про хорошу продуктивність моделі для цих об'єктів. Натомість для таких класів, як "tramway" і "person", криві менш стабільні, що може вказувати на труднощі моделі в їх передбаченні. Синя товста лінія представляє середнє значення F1 для всіх класів і відображає загальну продуктивність моделі. Згідно з графіком, найкраще значення F1 для всієї моделі становить 0.81 і досягається при порозі впевненості 0.320. Це оптимальний поріг, який забезпечує найкращий баланс між precision і recall.

Значення F1 для деяких класів різко знижується при високих порогах впевненості. Це вказує на те, що модель упускає частину об'єктів (низький recall), коли вимагається дуже висока впевненість. Водночас розкид між кривими для різних класів свідчить про різну продуктивність моделі. Для класів із нерівними

кривими або низькими піковими значеннями F1 може знадобитися додатковий аналіз, зокрема перевірка якості даних або їх збалансованості.

Цей графік дозволяє зробити висновок, що модель демонструє загалом хорошу продуктивність із найкращим F1 при порозі впевненості 0.320. Проте класи з нижчими значеннями F1, як-от "tramway" і "person", вимагають додаткової уваги. Можливо, потрібне доопрацювання датасету, збільшення кількості зображень цих класів або налаштування порогів confidence для окремих класів. Таким чином, графік F1-Confidence Curve є важливим інструментом для оцінки ефективності моделі, що дозволяє ідентифікувати сильні та слабкі сторони в її роботі.

Графік на рис. 3.6. демонструє ключові метрики і втрати (losses) під час тренування та валідації кастомної моделі. Він складається з кількох підграфіків, які відображають динаміку тренування моделі протягом епох. Графіки втрат, такі як train/box_loss і val/box_loss, показують поступове зниження втрат, пов'язаних із точністю визначення обмежувальних рамок об'єктів (bounding boxes). Це вказує на те, що модель поступово стає точнішою у визначенні меж об'єктів. Аналогічно, зменшення втрат train/cls_loss і val/cls_loss свідчить про покращення здатності моделі правильно класифікувати об'єкти. Втрати train/dfl_loss і val/dfl_loss, які пов'язані з передбаченням розподілу об'єктів, також зменшуються, що є позитивним сигналом про ефективність навчання.

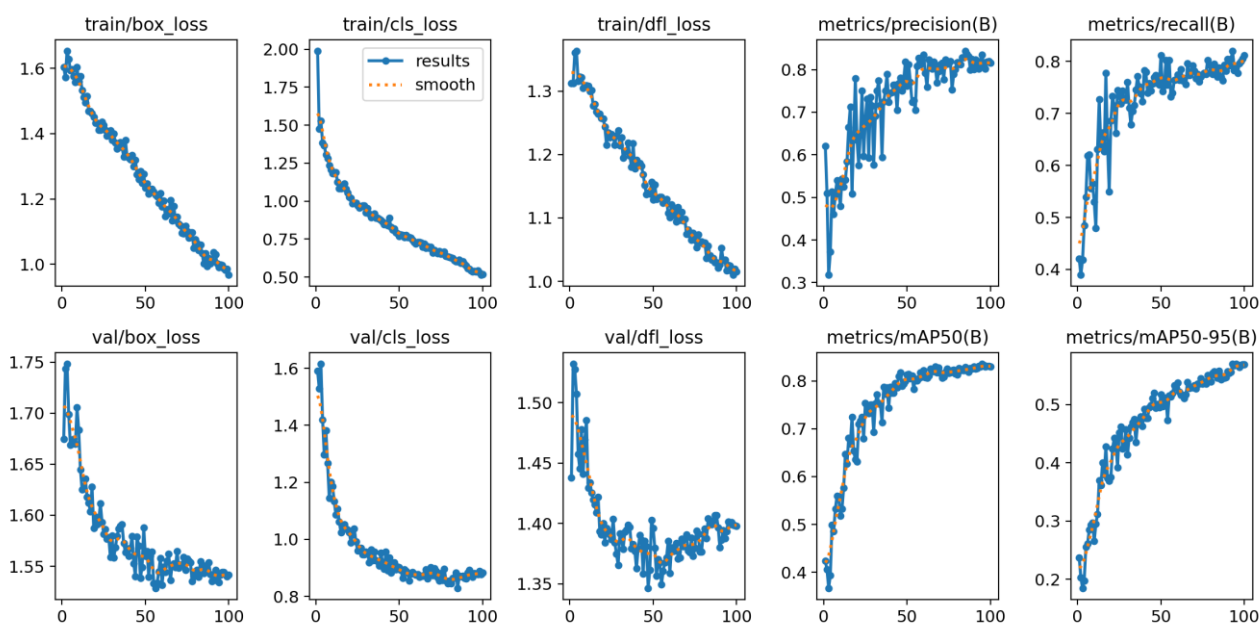


Рис. 3.6. Результат впродовж 100 епох навчання

На графіках метрик продуктивності видно поступове зростання значень. Графік `metrics/precision(B)` відображає покращення точності моделі, тобто її здатності правильно передбачати належність об'єкта до певного класу. Графік `metrics/recall(B)` показує, що модель стає краще у виявленні всіх об'єктів певного класу. Метрики `metrics/mAP50(B)` і `metrics/mAP50-95(B)` демонструють середню точність моделі для різних порогів IoU (Intersection over Union). Зростання mAP50 і mAP50-95 свідчить про те, що модель поступово стає більш ефективною у розпізнаванні об'єктів незалежно від класу.

Загалом, графік вказує на ефективне навчання моделі, оскільки втрати для тренувального і валідаційного наборів мають схожу тенденцію до зниження, що свідчить про відсутність явного перенавчання. Зростання метрик `precision`, `recall` і `mAP` підтверджує, що модель стає точнішою у розпізнаванні та класифікації об'єктів. Важливо, що значення mAP50-95 зростають, демонструючи, що модель здатна працювати з різними рівнями точності.

Результати цього графіка свідчать, що модель навчається добре, але важливо продовжувати моніторити динаміку втрат і метрик, щоб уникнути можливого перенавчання. Якщо метрики досягнуть плато, варто розглянути можливість

збільшення кількості епох, модифікації гіперпараметрів або поліпшення якості тренувального набору. Загалом, модель демонструє стабільне поліпшення, і її продуктивність поступово досягає високих показників точності та повноти.

На основі рис. 3.7. можна зробити висновок, що модель успішно виявляє транспортні засоби, такі як автомобілі, автобуси, мотоцикли та вантажівки. Для кожного зображення модель накладає боксові рамки на об'єкти, які розпізнаються, що свідчить про її базову функціональність. Однак помітно, що є певна кількість хибних виявлень та пропусків. Наприклад, деякі об'єкти ідентифіковані неправильно, особливо в умовах низького освітлення, а частина транспортних засобів залишилася нерозпізнаною. Модель працює як у денних умовах з хорошим освітленням, так і в умовах поганого освітлення, таких як нічні кадри. Проте якість детекції значно погіршується на нічних знімках через слабку видимість об'єктів. Це свідчить про те, що модель недостатньо навчена для роботи в складних умовах освітлення.

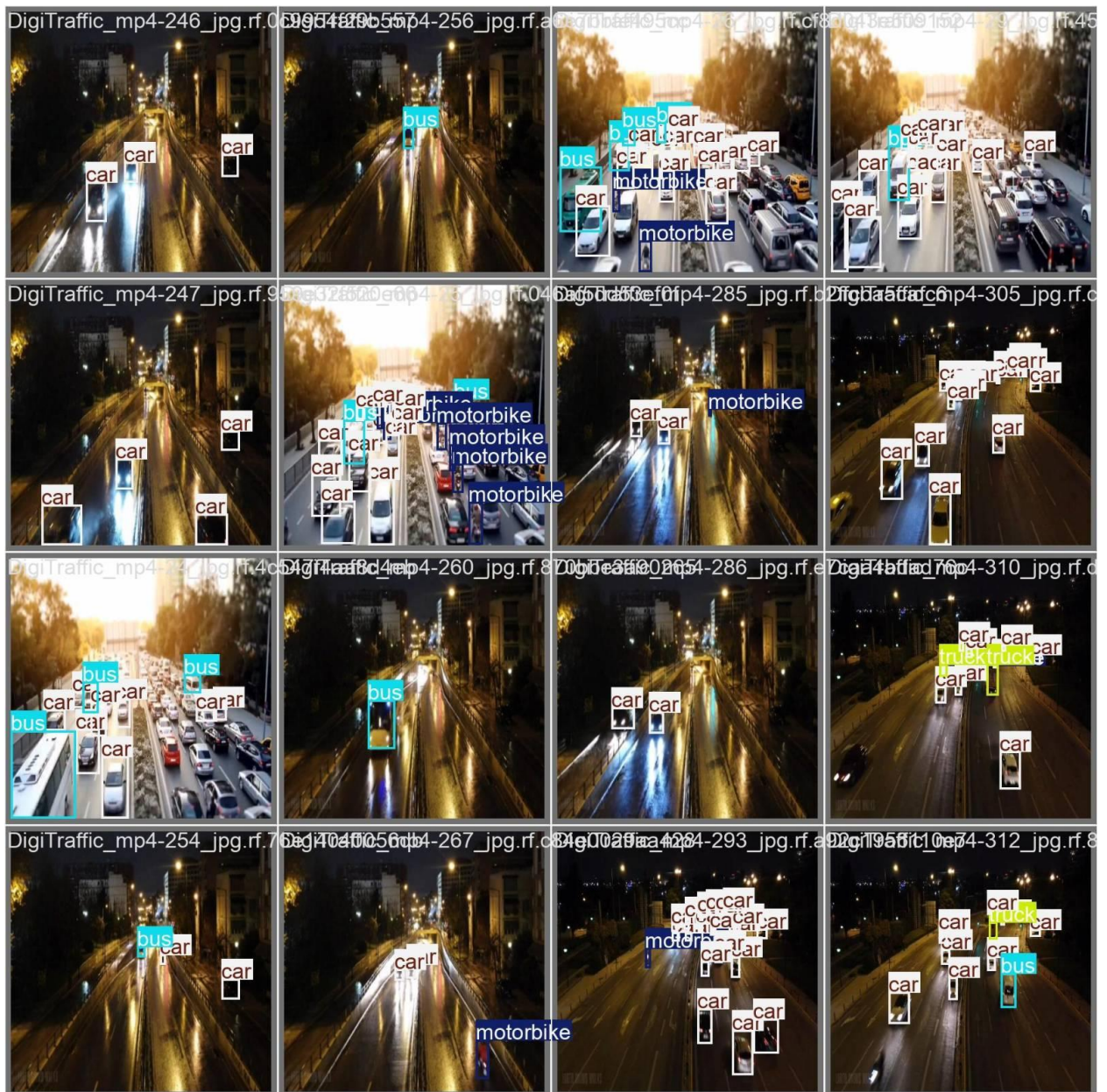


Рис. 3.7. Група зображень

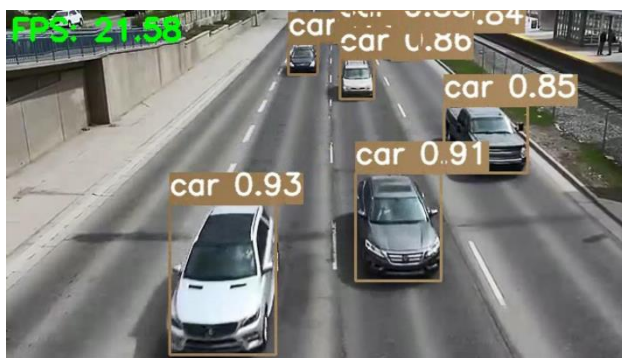
Результати виявлення об'єктів за допомогою різних моделей демонструють відмінності у швидкості та ефективності їхньої роботи рис.3.8. Для моделі ResNet-50 тривалість обробки відео становила 36 секунд, що значно перевищує час обробки іншими моделями. Хоча модель демонструє хорошу точність виявлення об'єктів, її низька швидкість обробки ($FPS = 21.58$) робить її менш придатною для застосувань у реальному часі.

Модель YOLOv8 значно швидша, із тривалістю обробки всього 15 секунд. Вона забезпечує високу швидкість ($FPS = 55.6$) при збереженні достатньої точності

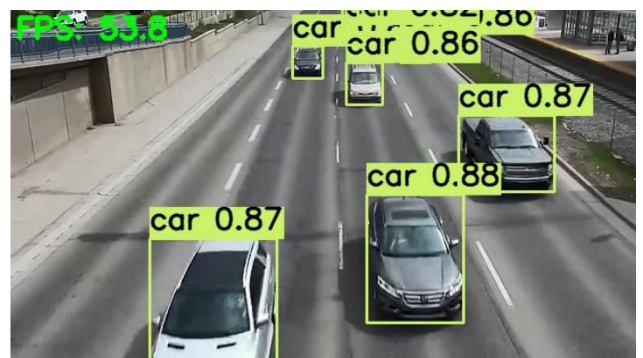
виявлення об'єктів, що робить її збалансованим вибором для задач реального часу. Однак можливі хибнопозитивні спрацювання можуть впливати на точність.

Користувальська модель, створена на основі YOLOv8, показала найвищу швидкість обробки, обробляючи відео за 13 секунд із $\text{FPS} = 58.32$. Крім того, ця модель змогла виявити більше об'єктів, ніж інші, проте середні рівні впевненості були дещо нижчими. Це свідчить про гарну адаптацію моделі до конкретного завдання, але вимагає подальшої оптимізації для підвищення точності та зниження кількості хибнопозитивних результатів.

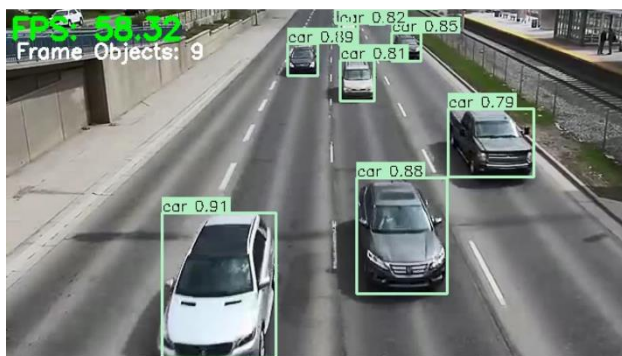
Слід зазначити, що тривалість основного відео становить 24 секунди, тому лише YOLOv8 та її користувальська версія здатні обробляти відео швидше, ніж його реальна тривалість. Це робить їх кращими варіантами для реальних застосувань, де потрібна обробка в режимі реального часу.



а



б



в

Рис. 3.8. Результати виявлення об'єктів, із застосуванням моделей: а) ResNet-50, б) YOLOv8, в) Користувальська модель на базі YOLOv8

На рис. 3.9. наведено результати виявлення об'єктів із застосуванням YOLOv8 у поєднанні з технологією SAHI. Видно, що модель змогла виявити 9 об'єктів, зокрема автомобілі та вантажівку, з різними рівнями впевненості, які варіюються від 0.51 до 0.86. Варто зазначити, що FPS становить лише 2.95, що значно нижче, ніж у попередніх тестах без застосування SAHI.

Загальний час обробки відео становив 5 хвилин 58 секунд, що є значно довшим, ніж у звичайної моделі YOLOv8 або її кастомної версії. Це свідчить про те, що використання SAHI, хоч і дозволяє виконувати детальніший аналіз зображень, значно знижує швидкість обробки, що може обмежити її застосування для задач реального часу.

Таким чином, модель YOLOv8 із SAHI підходить для випадків, де важлива точність і деталізація виявлення, але не критична швидкість обробки. Це може бути корисним для аналізу великих сцен із великою кількістю об'єктів або для складних задач, що вимагають високої деталізації.

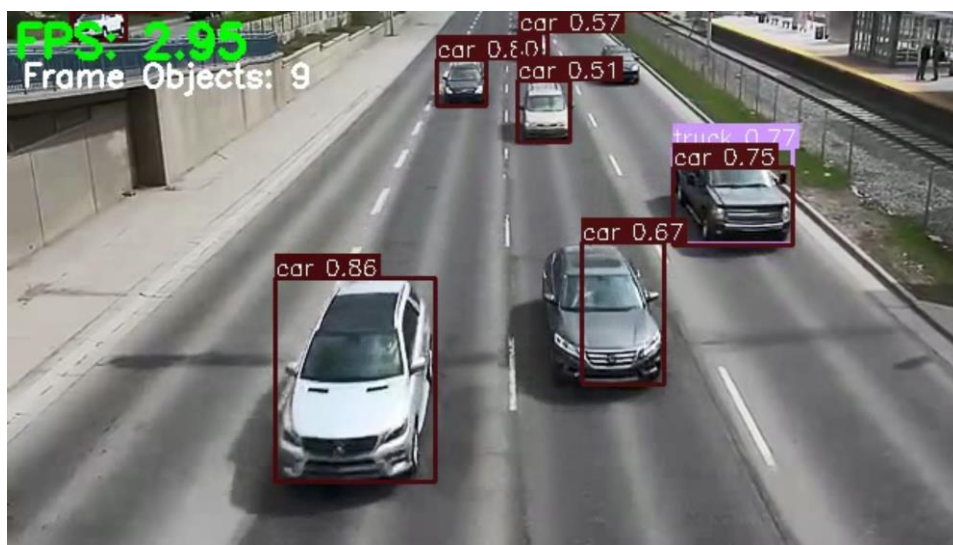


Рис. 3.9. Результати виявлення об'єктів, із застосуванням YOLOv8 + SAHI

На рис. 3.10.a показано результати роботи користувацької моделі на базі YOLOv8 із TensorRT. Видно, що модель виявила 9 об'єктів із рівнями впевненості в межах 0.55–0.91. FPS становить 110.04, що свідчить про дуже високу швидкість

обробки. Час обробки відео становив 7 секунд, що є значно швидше, ніж у базових моделей без оптимізації.

На рис. 3.10.б наведено результати роботи базової моделі YOLOv8 із TensorRT. Модель виявила 6 об'єктів із рівнями впевненості в межах 0.79–0.95. FPS у цьому випадку досягає 106.07, що також вказує на високу швидкість обробки. Загальний час обробки відео склав 5 секунд, що є найкращим результатом серед усіх тестованих моделей.

Таким чином, оптимізація моделей YOLOv8 за допомогою TensorRT значно підвищує швидкість обробки, роблячи їх придатними для задач у реальному часі. Користувацька модель показала більшу кількість виявлених об'єктів, що робить її ефективнішою для детальних аналізів, хоча базова модель демонструє трохи вищу точність і кращу швидкість.

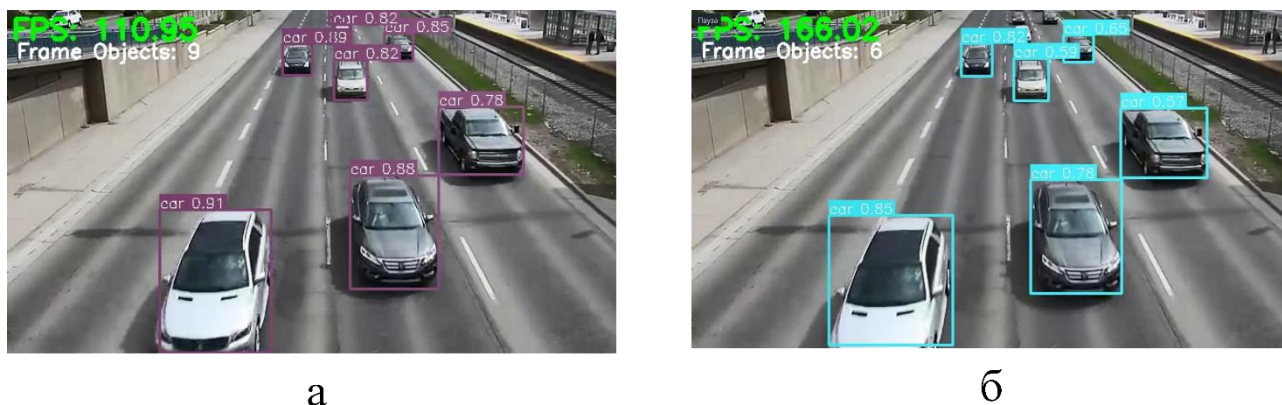
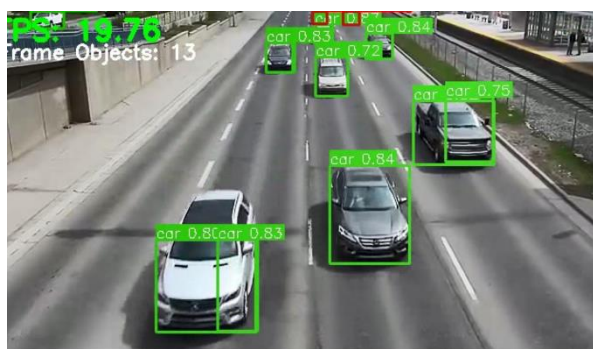


Рис. 3.10. Результати виявлення об'єктів, із застосуванням моделей: а) Користувацька модель на базі YOLOv8 + TensorRT, б) YOLOv8 + TensorRT

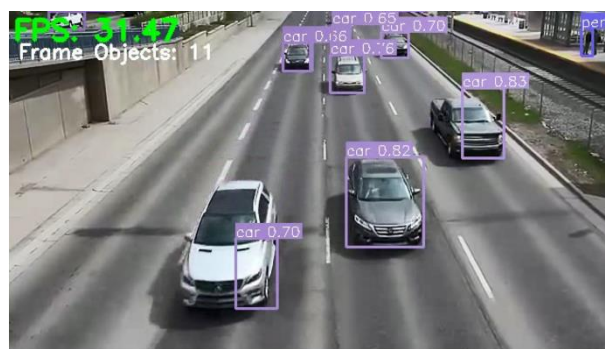
На зображенні 3.11.а показано результати роботи користувацької моделі на базі YOLOv8 із TensorRT і SAHI. Модель змогла виявити 13 об'єктів із рівнями впевненості, що варіюються від 0.54 до 0.85. FPS становить 13.78, що значно нижче порівняно з результатами моделей без SAHI. Загальний час обробки відео склав 34 секунди, що свідчить про те, що застосування SAHI значно впливає на швидкість обробки.

На зображенні 3.11.6 наведено результати роботи базової моделі YOLOv8 із TensorRT і SAHI. Модель виявила 11 об'єктів із рівнями впевненості в межах 0.56–0.78. FPS у цьому випадку дорівнює 31.67, що є кращим результатом порівняно з користувацькою моделлю. Загальний час обробки відео становив 23 секунди, що значно швидше.

Таким чином, додавання SAHI до моделей YOLOv8 із TensorRT дозволяє проводити детальніший аналіз, однак це значно знижує FPS і збільшує час обробки. Базова модель YOLOv8 демонструє кращий баланс між швидкістю та якістю, тоді як користувацька модель виявляє більше об'єктів, але збільшена кількість виявлених об'єктів є дублюючими.



а



б

Рис. 3.11. Результати виявлення об'єктів, із застосуванням моделей: а)Користувацька модель на базі YOLOv8 + TensorRT + SAHI , б) YOLOv8 + TensorRT + SAHI

Висновок до третього розділу

У третьому розділі було детально розглянуто повний цикл розробки та впровадження кастомної моделі розпізнавання об'єктів на платформі Jetson Nano. Починаючи з налаштування апаратної та програмної складової, було проілюстровано процес підготовки операційної системи, встановлення необхідних бібліотек та інструментів для глибинного навчання й обробки зображень.

Далі було описано процес створення власного набору даних за допомогою Roboflow, включаючи анотації зображень та застосування аугментацій, що дозволило розширити варіативність навчальної вибірки та покращити здатність моделі до узагальнення. Використання кастомних наборів даних спільно з попередньо навченими вагами YOLOv8 забезпечило ефективне донавчання моделей, які стали спеціалізованими на розпізнаванні конкретних об'єктів (наприклад, автомобілів).

Запуск процесу навчання у Google Colab продемонстрував гнучкість та зручність хмарних обчислень, дозволяючи швидко та ефективно отримати натреновану модель без значних локальних ресурсів. Було розглянуто методи оптимізації, зокрема, використання TensorRT для прискорення висновку моделі на Jetson Nano. Це значно підвищило швидкодію системи, роблячи її придатною для застосувань у реальному часі.

Також було проведено порівняльний аналіз продуктивності різних моделей і конфігурацій, включаючи поєднання YOLOv8 із SAHI, ResNet-50 та кастомної YOLOv8 із TensorRT. Отримані результати свідчать про важливість балансу між швидкістю та точністю, а також дають змогу вибирати оптимальну конфігурацію залежно від вимог конкретного застосування. Кастомна модель YOLOv8 із оптимізацією TensorRT показала високу швидкість та гнучкість, хоча за певних умов може знадобитися доопрацювання для підвищення точності.