Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра системного аналізу та інформаційних технологій

**Бакалаврська кваліфікаційна робота на тему:**

# «НЕЙРОМЕРЕЖЕВА ІНФОРМАЦІЙНА СИСТЕМА ВИЯВЛЕННЯ КУРЦІВ НА ОСНОВІ ПОТОКОВОГО ВІДЕО»

Виконав: студент 4 курсу, групи 2ІСТ-21б
спеціальності 126 – «Інформаційні системи та технології»
_____ Артем ЗНАМІРОВСЬКИЙ
Керівник: PhD., асистент. каф. САІТ
_____ Арсен ЛОСЕНКО
«04» 06 2025 р.
Рецензент: к.т.н., доц. каф. КН
_____ Олексій СІЛАГІН
«08» 06 2025 р.

**Допущено до захисту**

Завідувач кафедри САІТ
_____ д.т.н., проф. Віталій МОКІН
«06» 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра системного аналізу та інформаційних технологій
Рівень вищої освіти – перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність 126 – «Інформаційні системи та технології»
Освітньо-професійна програма – Прикладні інформаційні технології

ЗАТВЕРДЖУЮ
Завідувач кафедри САІТ
д.т.н., проф. Віталій МОКІН
"28" 03 2025 року

**ЗАВДАННЯ**
**НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ**
Знаміровському Артему Руслановичу

1. Тема роботи: «Нейромережева інформаційна система виявлення курців на основі потокового відео»
керівник роботи: Лосенко А.В., PhD., асистент каф. САІТ
затверджені наказом ВНТУ від «30» 03 2025 року № 97
2. Термін подання студентом роботи 31.05.25
3. Вихідні дані до роботи:
1) Bakhytov Y., Öz C. Cigarette Detection in Images Based on YOLOv8 Sakarya University Journal of Computer and Information Sciences. 2024. Vol. 7. DOI: https://doi.org/10.35377/saucis.2024.7.1.1461268
2) Датасет зображень курців, зібраний на платформі Roboflow: VNTU_Object_Detection_YOLO8s
3. Зміст звіту (перелік питань, які потрібно розробити):
1) Аналіз проблематики задачі виявлення курців.
2) Вибір оптимальної моделі та технологій для розв'язання поставленої задачі.
3) Опис процесу розроблення інформаційної системи.
4) Результат застосування технології на реальних даних
4. Зміст текстової частини:
1) Аналіз проблематики задачі виявлення курців
2) Вибір оптимальних інформаційних технологій.
3) Опис процесу розроблення інформаційної системи
5. Перелік ілюстративного матеріалу:
1) Діаграма використання мобільного застосунку;
2) Архітектура інформаційної системи;
3) Діаграма класів інформаційної системи;
4) UML-діаграма діяльності мобільного застосунку;

5) Інтерфейс мобільного застосунку;
6) Інтерфейс вебклієнту;
7) Робота нейромережі у вебінтерфейсі.

6. Консультанти розділів роботи:

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання 28.03.25

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва та зміст етапу | Термін виконання | | Примітка |
|---|---|---|---|---|
| | | початок | закінчення | |
| 1 | Аналіз предметної області | 01.04 | 10.04 | вик |
| 2 | Порівняльний аналіз проблематики | 10.04 | 20.04 | вик |
| 3 | Вибір оптимальних інформаційних технологій | 20.04 | 30.04 | вик |
| 4 | Розроблення інформаційної системи виявлення курців на основі потокового відео | 1.05 | 15.05 | вик |
| 5 | Оформлення матеріалів до захисту БКР | 15.05 | 30.05 | вик |

Студент _____ Артем ЗНАМІРОВСЬКИЙ

Керівник роботи _____ Арсен ЛОСЕНКО

# АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 75 сторінки формату А4, на яких є 47 рисунків, список використаних джерел містить 27 найменувань.

Метою роботи є розроблення інформаційної системи для виявлення курців у режимі реального часу на основі аналізу потокового відео. Для навчання моделі створено спеціалізований датасет на платформі Roboflow. Проведено експерименти з використанням моделей YOLO та Roboflow. Найефективнішу модель було інтегровано у мобільний додаток, реалізований мовою Kotlin. Система дозволяє автоматично виявляти випадки куріння у громадських місцях, що може використовуватись у цілях громадського контролю та безпеки.

Ключові слова: виявлення курців, комп’ютерний зір, потокове відео, YOLO, Roboflow, Kotlin, мобільний додаток.

# ABSTRACT

The bachelor's thesis consists of 75 A4 pages, which contain 47 figures, the list of sources used contains 27 titles.

The aim of the work is to develop an information system for real-time smoker detection from streaming video.

A specialized dataset was created on the Roboflow platform, and multiple object detection models, including YOLO and Roboflow, were evaluated. The best-performing model was integrated into a mobile application developed in Kotlin. The system enables automated identification of smoking events in public spaces and can be used for public safety and surveillance purposes.



Keywords: smoker detection, computer vision, video stream, YOLO, Roboflow, Kotlin, mobile application.

# ЗМІСТ

# ВСТУП

**Актуальність теми**. Сучасні інформаційні технології дедалі частіше використовуються для вирішення завдань громадської безпеки та моніторингу порушень у публічному просторі. Однією з актуальних проблем, що потребує вирішення, є порушення законодавства щодо заборони куріння у громадських місцях. Куріння, зокрема серед молоді, залишається поширеним явищем, що негативно впливає як на здоров’я самих курців, так і на людей навколо.

У зв’язку з цим зростає потреба у створенні автоматизованих систем, здатних виявляти факти куріння у режимі реального часу без участі людини. Використання технологій комп’ютерного зору, глибинного навчання та мобільних пристроїв відкриває нові можливості для реалізації таких рішень. Інформаційні системи, засновані на нейронних мережах, дозволяють здійснювати обробку відеопотоку безпосередньо на пристрої, фіксувати порушення та забезпечувати формування відповідної аналітики.

Таким чином, розробка нейромережевої інформаційної системи для виявлення куріння на основі потокового відео є своєчасною та актуальною задачею. Її реалізація сприятиме підвищенню ефективності контролю за дотриманням норм поведінки у громадських місцях, формуванню здорового середовища, а також забезпечить практичне застосування сучасних технологій штучного інтелекту в реальних умовах.

**Мета і задачі дослідження.** Метою дослідження є розроблення інформаційної системи виявлення курців у потоковому відео з використанням нейромережевої моделі виявлення об’єктів.

Для досягнення поставленої мети були поставлені наступні задачі:

– провести аналіз проблематики виявлення курців у потоковому відео;

– здійснити вибір оптимальних інформаційних технологій та платформ для розробки програмного забезпечення і взаємодії з діючою серверною частиною;

– здійснити розробку мобільного та веб-додатку для роботи клієнтської частини.

**Об’єктом дослідження** є нейромережева інформаційна система для виявлення куріння у режимі реального часу на основі аналізу потокового відео.

**Предметом дослідження** є методи та програмні засоби розробки мобільної інформаційної системи, що здійснює автоматичне виявлення фактів куріння з використанням технологій комп’ютерного зору та глибинного навчання.

**Публікації.** За результатами даної роботи була зроблена доповідь на тему «Нейромережева інформаційна система виявлення курців на основі потокового відео» на Міжнародній науково-практичній інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи» (2025) з публікацією тез.

# 1 АНАЛІЗ ПРОБЛЕМАТИКИ ЗАДАЧІ ВИЯВЛЕННЯ КУРЦІВ

## 1.1 Аналіз проблематики тютюнопаління та шляхи її інформаційного контролю

Тютюнопаління є однією з найсерйозніших проблем громадського здоров'я в Україні. За даними МОЗ, щороку від захворювань, пов'язаних із курінням, помирає близько 130 тисяч українців. Станом на 2023 рік, 27,4% дорослого населення України є курцями (рис. 1.1), з них 44% — чоловіки та 13,7% — жінки [1]. Хоча загальна тенденція свідчить про зменшення поширеності куріння — з 30% у 2023 році до 27% у 2024 році, проблема залишається актуальною, особливо серед молоді та вразливих груп населення [2].


Рисунок 1.1 — Інфографіка віку споживачів тютюнових та нікотинових виробів

Особливу тривогу викликає підліткове куріння: понад 20 % дітей віком 13–15 років мають досвід куріння. Хоча на рис. 1.2 вказано, що поширеність куріння серед підлітків скоротилася від 24 % у 2005 до 9,2 % у 2017, у 2023 вона виросла

до приблизно 12,3 % і закріпилася на цьому рівні протягом останніх семи років [3].

Електронні сигарети та вейпи стали ще доступнішими і популярнішими серед школярів, що сприяє ранньому формуванню нікотинової залежності — тенденція, яка вимагає оперативних і ефективних заходів [4].

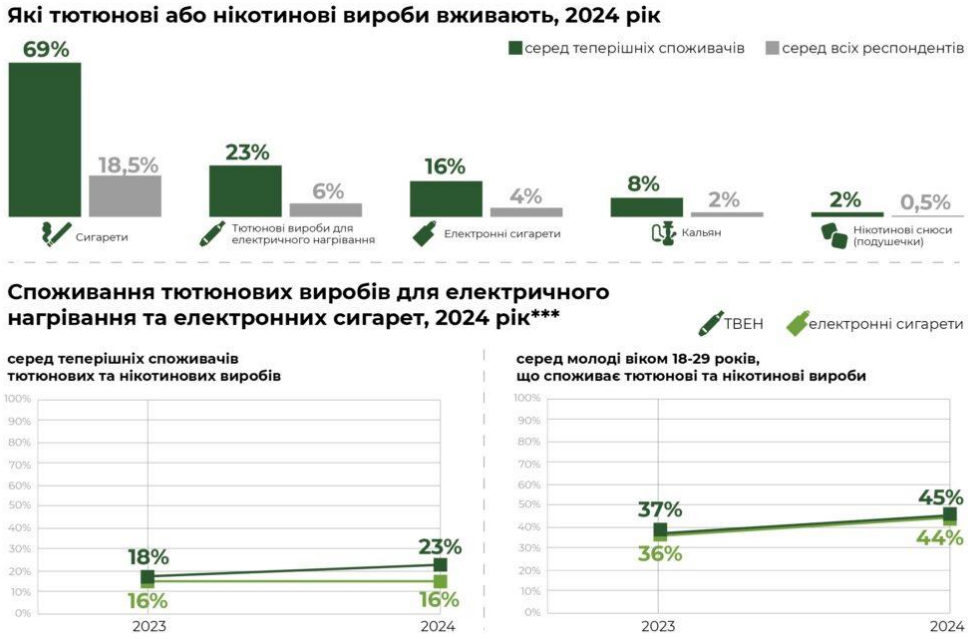


Рисунок 1.2 — Інфографіка вживання тютнових та тютюнових виробів

У відповідь на проблему виникла потреба у впровадженні інноваційних інформаційних систем для виявлення курців у режимі реального часу. Одним із перспективних підходів є розробка систем на основі нейромереж, які можуть автоматично ідентифікувати факти куріння у потоковому відео. Це дає змогу ефективніше контролювати дотримання правил у громадських місцях і запобігати шкідливим звичкам серед молоді.

## 1.2 Технології сучасних засобів для куріння та їх розпізнавання

Сучасний ринок тютюнових і нікотинових виробів постійно розширюється, що ускладнює боротьбу з курінням у громадських місцях. Окрім традиційних сигарет, які залишаються поширеним джерелом тютюнової залежності, дедалі популярнішими стають альтернативні пристрої для вживання

нікотину, зокрема електронні сигарети, вейпи, системи нагрівання тютюну та под-системи.

Класичні тютюнові вироби (рис 1.3), як-от сигарети та сигари, є легко впізнаваними, однак вимагають врахування змін у вигляді кольору диму, наявності вогника та характерної позиції руки користувача під час паління.

Рисунок 1.3 – Візуальні ознаки традиційного куріння: дим, вогник, положення руки

Електронні сигарети (рис. 1.4) поділяються на декілька видів: картриджні системи (наприклад, JUUL), системи з вільним заправленням (вейпи з резервуаром), под-системи (одноразові або багаторазові пристрої у формі флешки).

Ці пристрої створюють аерозоль, а не дим, мають мінімальний запах і часто виглядають як невеликі електронні пристрої, що ускладнює їхнє виявлення у відеопотоці.

Рисунок 1.4 – Основні типи електронних сигарет та вейпів

Системи нагрівання тютюну (наприклад, IQOS, glo) не створюють відкритого полум’я, не дають стійкого диму, а процес куріння триває короткий проміжок часу (рис 1.5).

Рисунок 1.5 – Пристрій для нагрівання тютюну

Наявність великої кількості варіантів курильних пристроїв значно ускладнює створення класичної системи контролю, побудованої на простому аналізі зображення чи відео. Саме тому виявлення курців у режимі реального часу потребує використання глибинного навчання, зокрема нейромережевих моделей розпізнавання об'єктів, які здатні враховувати різноманітні форми, кольори, пози і контексти використання тютюнових виробів (рис. 1.6).

Рисунок 1.6 – Різні типи куріння

Виявлення куріння в сучасних умовах є складною багатофакторною задачею, яка потребує адаптації систем до різних типів курильних пристроїв, що відрізняються зовнішнім виглядом, механізмом дії та поведінкою користувача.

## 1.3 Переваги та виклики використання нейромережевих систем виявлення куріння

Переваги:

1. Автоматизація моніторингу громадських місць. Нейромережеві системи здатні виявляти куріння у відеопотоці в режимі реального часу, що дозволяє зменшити навантаження на персонал охорони та забезпечити постійний контроль у місцях з великою кількістю людей (рис. 1.7).


Рисунок 1.7 – Приклад відеоконтролю у громадському місці

2. Висока точність при навчанні на релевантних даних. При належному навчанні на зображеннях сигарет, вейпів і курців у різних умовах, нейромережі демонструють високу ефективність у виявленні заборонених дій.

3. Можливість інтеграції у мобільні й десктопні системи. Сучасні моделі можуть бути адаптовані для роботи на різних пристроях — від камер відеоспостереження до мобільних застосунків, що відкриває широкі можливості для використання.

4. Підвищення рівня профілактики та реагування. Завдяки швидкому виявленню куріння система може оперативно сповіщати відповідальні служби, фіксувати факти порушень та допомагати у прийнятті превентивних заходів.

5. Збереження доказів. Такі системи здатні фіксувати момент куріння (зображення/відео), зберігати їх з мітками часу та формувати аналітичні або юридичні звіти.

Виклики:

1. Складність у виявленні різних форм куріння. Куріння електронних сигарет або вейпів має менше візуальних ознак (наприклад, слабший дим), тому нейромережі можуть не завжди точно розпізнати акт куріння.

2. Чутливість до умов зйомки. Освітлення, якість відео, ракурси — все це впливає на якість виявлення. Наприклад, вночі або в умовах слабкого освітлення точність детекції може знижуватись.

3. Потреба в обширному та збалансованому датасеті. Для якісної роботи модель потрібно навчити на великій кількості даних, що включають різні типи курців, позицій, фонів та умов середовища.

4. Ризик хибних спрацьовувань. Можливі ситуації, коли система приймає безпечні дії за куріння, що може викликати непорозуміння або потребу в перевірці вручну.

5. Етичні та правові аспекти. Впровадження систем відеоаналітики передбачає збір і обробку персональних даних, що потребує дотримання законодавства про конфіденційність.

## 1.4 Аналіз аналогічних методів та інформаційних систем

У сучасних інформаційних системах контролю поведінки активно застосовуються технології комп’ютерного зору та штучного інтелекту для виявлення шкідливих звичок, таких як куріння. Основним підходом є використання глибоких нейронних мереж, зокрема архітектур сімейства YOLO (You Only Look Once), які забезпечують високу швидкодію та точність виявлення об'єктів у реальному часі [5].

Одним із прикладів є алгоритм YOLOv8-MNC, який розширює базову модель YOLOv8 (рис. 1.8) шляхом додавання спеціалізованого шару для виявлення дрібних об'єктів, таких як сигарети [6].

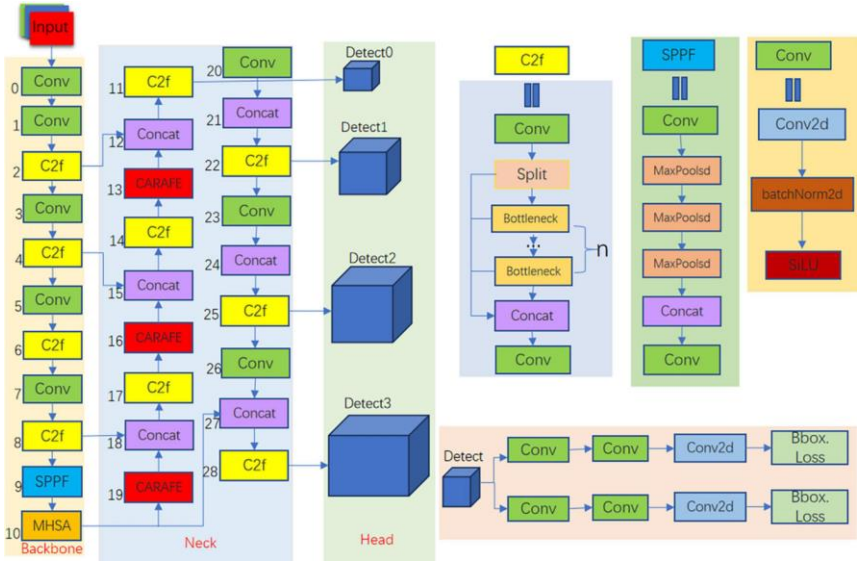


Рисунок 1.8 — Структура YOLOv8-MNC

Цей підхід також включає механізм багатоголової самоуваги (MHSA) та легкий оператор апсемплінгу CARAFE (рис. 1.9), що дозволяє досягти точності виявлення 85,89% на спеціалізованому датасеті та значно розширити функціонал модулів завдяки розширеним конфігураціям (рис. 1.10).

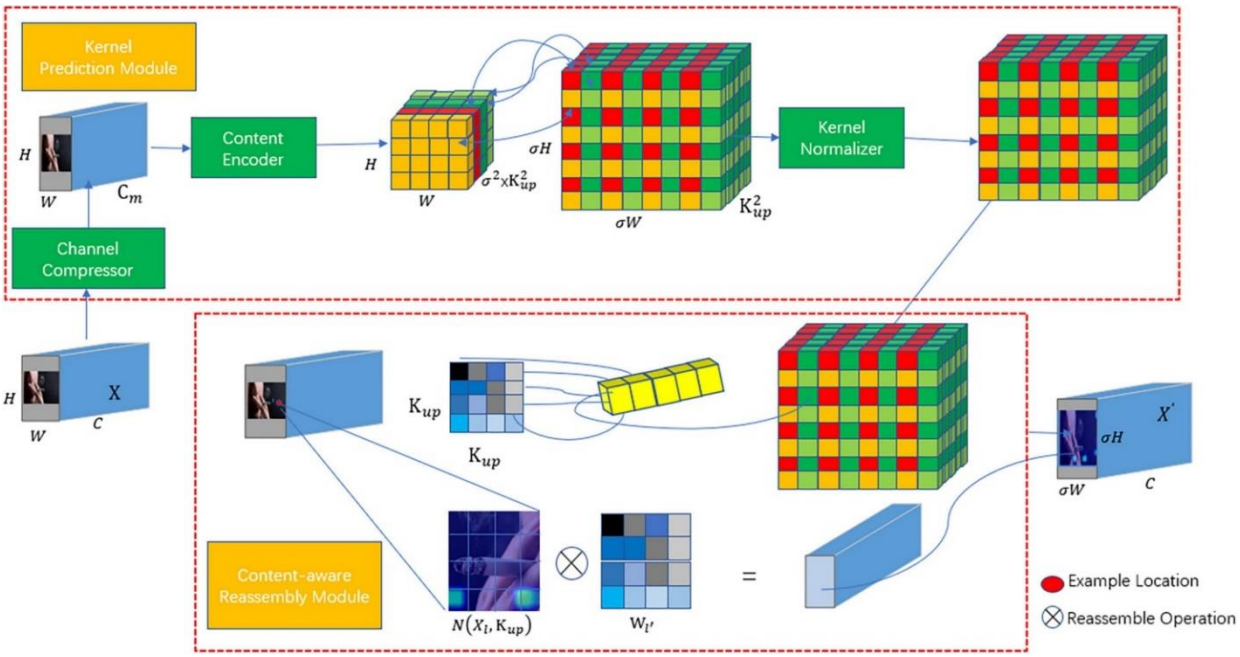


Рисунок 1.9 — Схема роботи модуля CARAFE

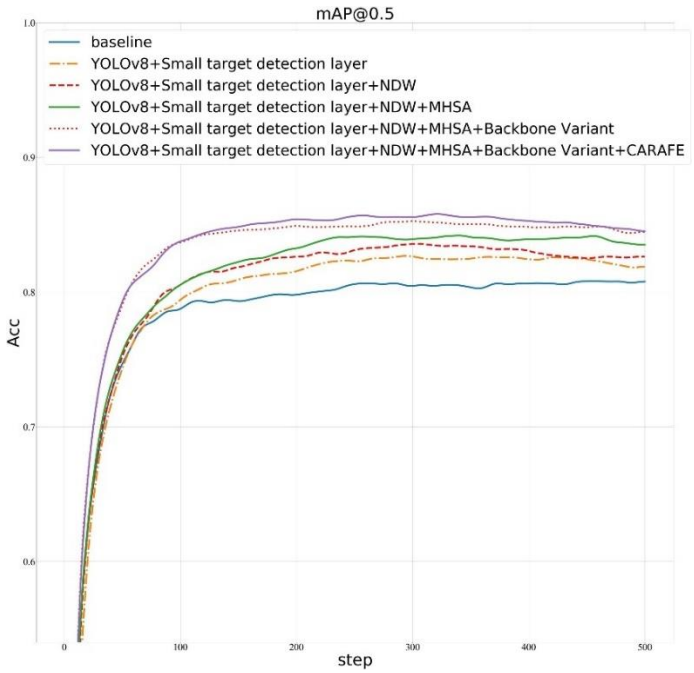


Рисунок 1.10 — Графік результатів абляційного експерименту для різних конфігурацій моделі YOLOv8

На рисунку 1.11 представлено порівняння матриць плутанини для моделей YOLOv8 та YOLOv8-MNC, яке демонструє поліпшення точності класифікації класу «smoke» з 0.79 до 0.83 та зменшення частки хибнонегативних передбачень із 0.21 до 0.17 відповідно. Це свідчить про підвищену здатність модифікованої моделі YOLOv8-MNC ефективніше виявляти дим, що підтверджує доцільність застосування архітектурних покращень у задачах виявлення куріння(рис. 1.12).

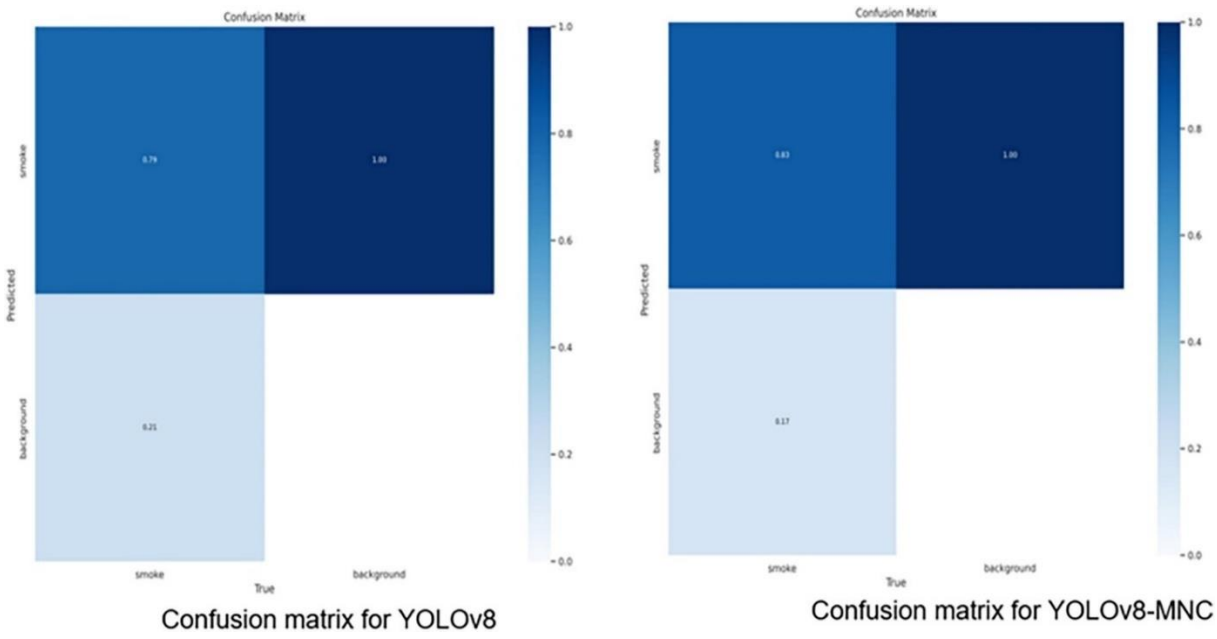


Рисунок 1.11 — Порівняння матриць помилок для моделей YOLOv8 та YOLOv8-MNC


Рисунок 1.12 — Приклад роботи YOLOv8-MNC

Крім того, існують практичні реалізації систем виявлення куріння, зокрема:

"Detect People Smoking" (Rui Santos) - Kaggle:

Цей проєкт реалізує підхід до бінарної класифікації зображень, спрямований на виявлення факту куріння. Модель визначає, чи містить зображення курця або сигарету, однак не виконує локалізацію об’єктів (рис. 1.13) [7].

Основні етапи реалізації включають використання попередньо зібраного датасету з зображеннями куріння та некуріння, тренування класифікаційної моделі на основі згорткової нейронної мережі (CNN), а також побудову метрик ефективності моделі, зокрема точності, повноти та кривої ROC.


Рисунок 1.13 — Приклад роботи "Detect People Smoking"

На рисунку 1.14 зображено нормалізовану матрицю плутанини для моделі бінарної класифікації (класи “Yes” — наявність куріння, “No” — відсутність). Значення в комірках подано у вигляді частки від загальної кількості прикладів

відповідного класу, що дозволяє оцінити якість класифікації незалежно від дисбалансу в даних.

Верхній лівий елемент (0.97) відповідає частці правильно класифікованих прикладів класу “No” (істинно негативних), тобто випадків, коли модель правильно розпізнала відсутність куріння. Верхній правий елемент (0.03) — це хибнопозитивні передбачення, коли модель помилково вказала на наявність куріння там, де його не було.

Нижній лівий елемент (0.14) демонструє хибнонегативні випадки — ситуації, коли система не змогла виявити куріння, хоча воно було присутнє. Нижній правий елемент (0.85) — істиннопозитивні результати, тобто правильне виявлення куріння.

Такий розподіл свідчить про високу точність виявлення негативного класу (відсутності куріння), а також про добрий, хоча й не ідеальний рівень коректного виявлення позитивного класу. Отже, модель демонструє збалансовану продуктивність, але має простір для покращення в напрямку зменшення кількості хибнонегативних передбачень.

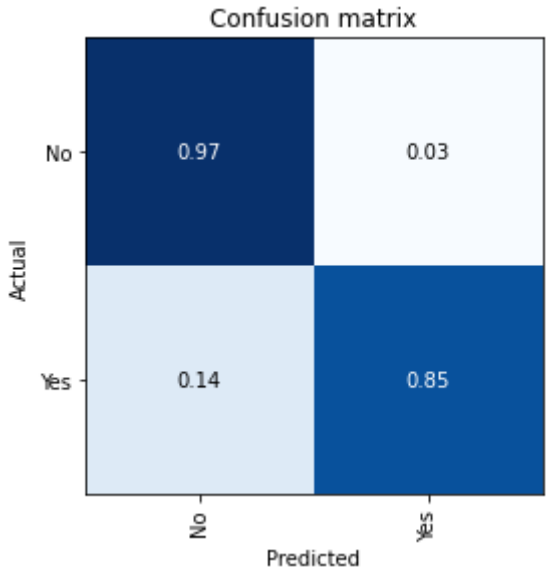


Рисунок 1.14 — Абсолютна матриця помилок моделі

Перевага цього підходу — простота реалізації. Проте відсутність локалізації об'єктів обмежує можливості використання в системах відеоспостереження чи правової фіксації порушень.

"Smoking Detection – Accuracy 89.6%" (Abdallah Wagih) - Kaggle: У цьому рішенні реалізовано підхід до класифікації зображень на основі згорткової нейронної мережі (CNN), яка не виконує об’єктної детекції, а лише визначає факт наявності куріння на зображенні [8]. Ключовими характеристиками є обробка набору статичних зображень, навчання моделі CNN для бінарної класифікації («куріння» / «без куріння»), а також обчислення стандартних метрик ефективності, таких як точність, повнота та середня точність (рис. 1.15).

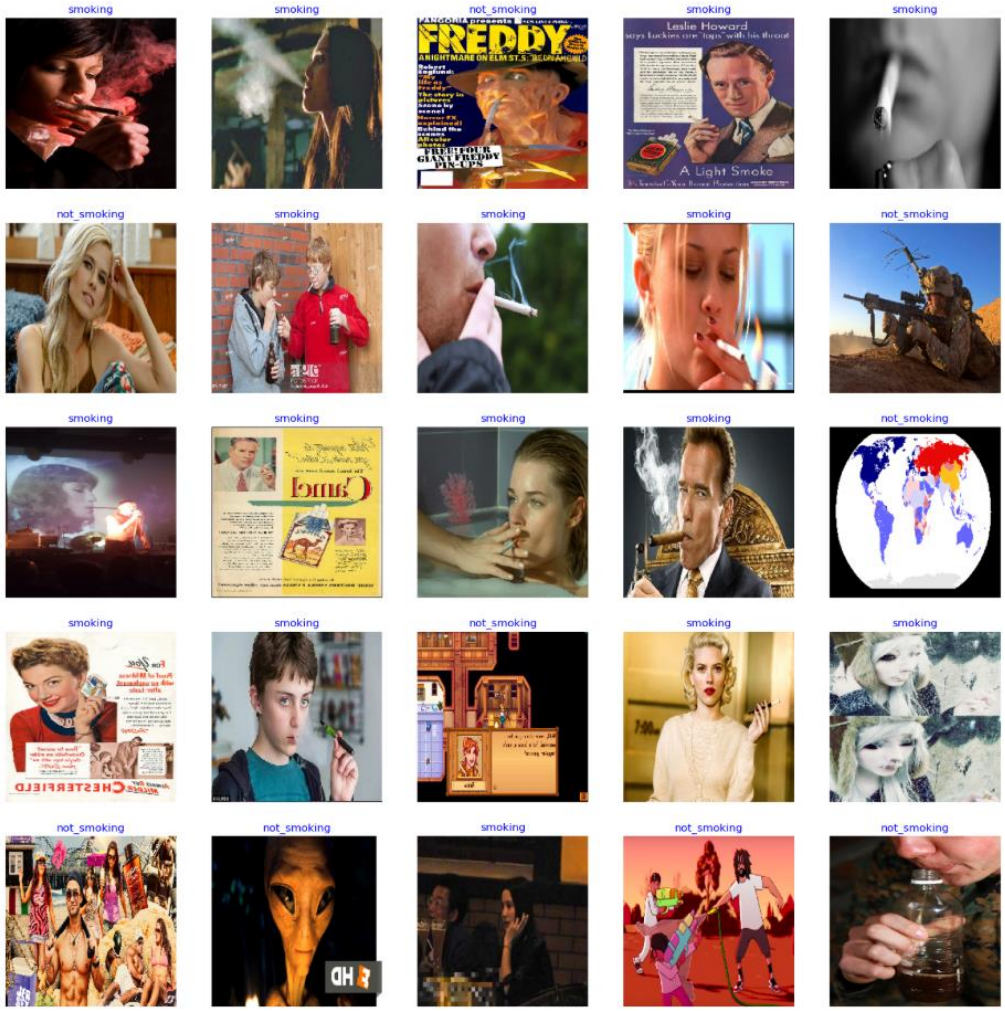


Рисунок 1.15 — Приклад роботи "Smoking Detection – Accuracy 89.6%" (Abdallah Wagih)

На рисунку 1.16 зображено графіки зміни функції втрат (Loss) та точності (Accuracy) для тренувальної та валідаційної вибірок протягом 15 епох навчання.

Графік зліва ілюструє зменшення функції втрат для обох вибірок. Червона лінія відображає значення Training Loss, зелена — Validation Loss. Зменшення

обох кривих свідчить про поступове навчання моделі. Найменше значення Validation Loss зафіксовано на 15-й епосі (позначено синьою точкою), що вказує на оптимальну здатність моделі до узагальнення саме на цьому етапі.

На правому графіку відображено зміну точності: Training Accuracy (червона лінія) демонструє майже лінійне зростання до значень, близьких до 1.0, тобто 100%. Це вказує на високу здатність моделі точно відтворювати тренувальні приклади. Водночас Validation Accuracy (зелена лінія) коливається в межах 0.86–0.89, що є типовим показником для задачі класифікації з урахуванням валідаційного шуму. Найкраща валідаційна точність досягнута на 13-й епосі (позначено синьою точкою), після чого спостерігається незначне погіршення.

Отже, найкраща модель за точністю — на 13-й епосі, а за мінімальними втратами — на 15-й. Такий аналіз дозволяє обрати найбільш збалансовану версію моделі відповідно до цільових метрик.

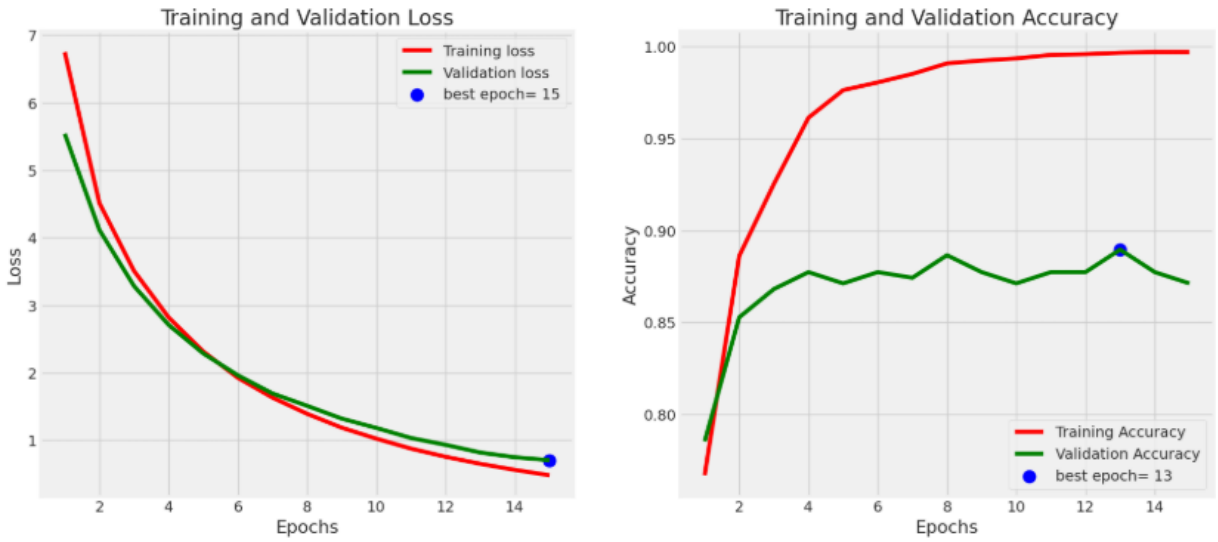


Рисунок 1.16 — графік  Training and Validation Loss

На рисунку 1.17 представлено матрицю помилок, що відображає результати класифікації зображень на два класи — *smoking* (куріння) та *not_smoking* (відсутність куріння). Згідно з нею, модель правильно класифікувала 182 зображення як куріння (True Positives) та 105 зображень як некуріння (True Negatives). Помилковими виявилися 21 передбачення куріння при його

фактичній відсутності (False Positives) і 12 випадків, коли система не розпізналанаявність куріння (False Negatives).

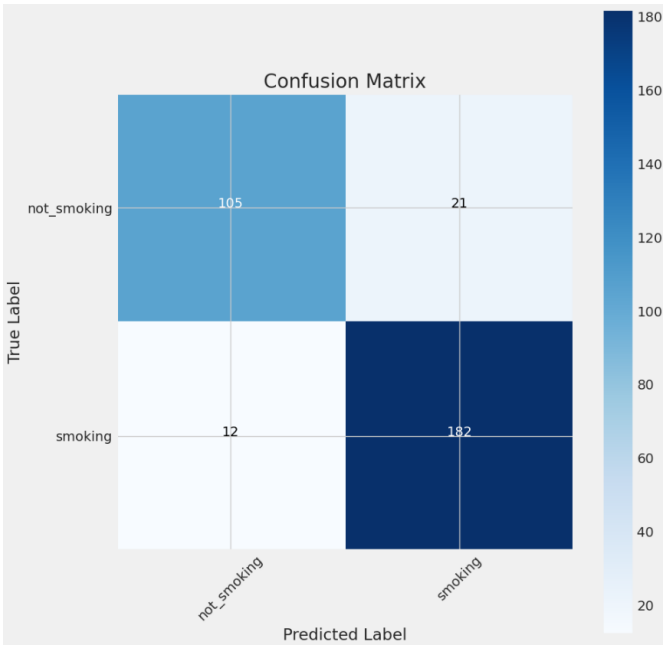


Рисунок 1.17 — абсолютна матриця помилок

Цей підхід демонструє хорошу якість класифікації, однак не дозволяє локалізувати об’єкти куріння, що обмежує його практичне використання в системах відеоспостереження.

Розглянуті проєкти демонструють ефективність застосування методів глибокого навчання для задачі виявлення куріння на зображеннях. Обидва рішення реалізовані у формі класифікаторів кадрів, де результатом є бінарне визначення наявності або відсутності ознак куріння. При цьому локалізація об'єктів, яка характерна для задач об'єктного детектування, не реалізується.

Проєкти досягають високої точності, що підтверджується графіками навчання, метриками precision, recall та матрицею помилок. Це свідчить про потенціал використання таких моделей у системах первинного фільтрування або оцінки ризику. Водночас обмеженням залишається відсутність просторової прив’язки виявлених об’єктів, що знижує практичну ефективність в умовах реального моніторингу.

Світовий досвід також підтверджує доцільність впровадження подібних рішень. Наприклад, у Китаї активно використовуються автоматизовані системи відеонагляду в навчальних закладах та на вокзалах для фіксації дисциплінарних

порушень, зокрема куріння. У США інтелектуальні камери застосовуються в університетах, торгових центрах та транспортних вузлах для виявлення диму, зброї та підозрілої поведінки.

Тому, використання нейромережевих моделей в інформаційних системах дозволяє підвищити ефективність моніторингу поведінки в публічному середовищі, зокрема у школах, навчальних закладах, офісах та на підприємствах, де важливо оперативно реагувати на порушення заборони тютюнопаління.

## 1.5 Висновки

У цьому розділі було здійснено комплексний аналіз проблеми тютюнопаління в Україні, з акцентом на сучасні виклики, пов’язані з поширенням електронних сигарет, вейпів і под-систем, зокрема серед підлітків та молоді.

Розглянуто сучасні типи тютюнових і нікотинових виробів, їх візуальні особливості та специфіку виявлення таких об’єктів у відеопотоці.

Обґрунтовано доцільність використання нейромережевих систем як основи для побудови сучасних інформаційних технологій виявлення куріння у режимі реального часу. Виокремлено ключові переваги таких систем: автоматизація моніторингу, висока точність за умови якісного навчання, можливість інтеграції у різні пристрої, оперативне реагування на порушення та збереження доказів

Також було проаналізовано два приклади реалізації моделей класифікації зображень із використанням нейромереж. Обидва проєкти демонструють високу точність визначення наявності ознак куріння на зображенні, проте не здійснюють локалізації об'єктів, що обмежує можливості їх застосування в системах відеоспостереження. Встановлено, що такі рішення можуть бути ефективними в системах попереднього фільтрування, однак не повністю покривають потребу в просторовому виявленні об’єктів для подальшого реагування.

Таким чином, результати огляду підтверджують доцільність застосування глибинного навчання у сфері автоматизованого моніторингу дотримання заборони куріння, а також окреслюють перспективи вдосконалення існуючих рішень шляхом поєднання класифікаційних моделей із алгоритмами виявлення об’єктів.

# 2 ВИБІР ОПТИМАЛЬНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

## 2.1 Аналіз можливостей мови програмування Kotlin

Kotlin — це сучасна мова програмування, створена компанією JetBrains у 2011 році, яка з 2017 року офіційно підтримується Google як основна мова розробки Android-застосунків. Її популярність зумовлена поєднанням лаконічного синтаксису, безпечної роботи з типами (null safety), підтримки функціонального та об'єктно-орієнтованого підходів, а також глибокої інтеграції з платформою Android [9].

У межах проєкту мобільної нейромережевої системи виявлення курців Kotlin став основною мовою реалізації. Його використання дозволило досягти як високої швидкості розробки, так і надійності програмної логіки (рис. 2.1).

Kotlin забезпечив ефективну роботу з бібліотеками Android Jetpack, зокрема такими як CameraX, ViewModel, LiveData, що дало змогу структурувати архітектуру застосунку згідно з сучасними підходами. Система nullable-типів дозволила уникнути багатьох помилок, пов’язаних із обробкою об’єктів, які можуть не містити значення, а механізм корутин значно спростив реалізацію асинхронної обробки даних — зокрема, запуску інференції моделі та роботи з відеопотоком без блокування основного потоку.

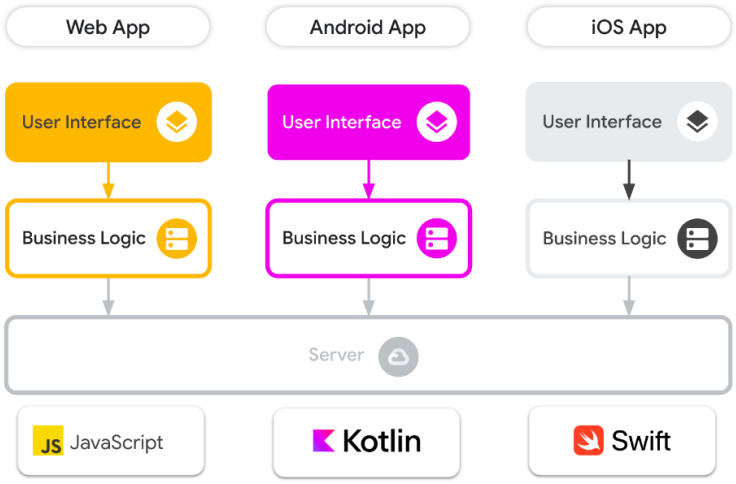


Рисунок 2.1 — Порівняння архітектури веб-, Android- та iOS-застосунків за рівнями логіки

У проєкті Kotlin використовуватиметься для реалізації захоплення відео в реальному часі через CameraX (рис. 2.2), інтеграції моделі TensorFlow Lite для локального виконання інференції, обробки кадрів (перетворення ImageProxy у Bitmap, масштабування, нормалізація) та передачі MJPEG-потоку у браузер із накладеними результатами виявлення [10]. Окрім того, мова забезпечуватиме реалізацію логування подій, збереження знімків із часовими мітками та генерацію PDF-звітів, що підсумовують зафіксовані порушення.

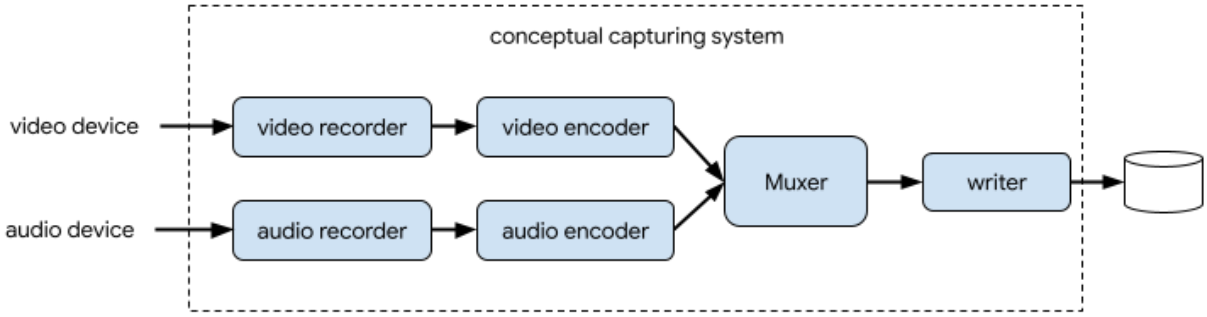


Рисунок 2.2 — Архітектура відео та аудіо захоплення Camera X

Висока продуктивність, сумісність із Java, безпечне управління пам’яттю та активна підтримка з боку спільноти зробили Kotlin ідеальним вибором для створення мобільної інформаційної системи реального часу (рис. 2.3). Його можливості дозволили реалізувати повноцінний застосунок, здатний працювати автономно на пристроях середнього рівня, забезпечуючи точне виявлення куріння, збереження результатів та інтуїтивну взаємодію з користувачем.

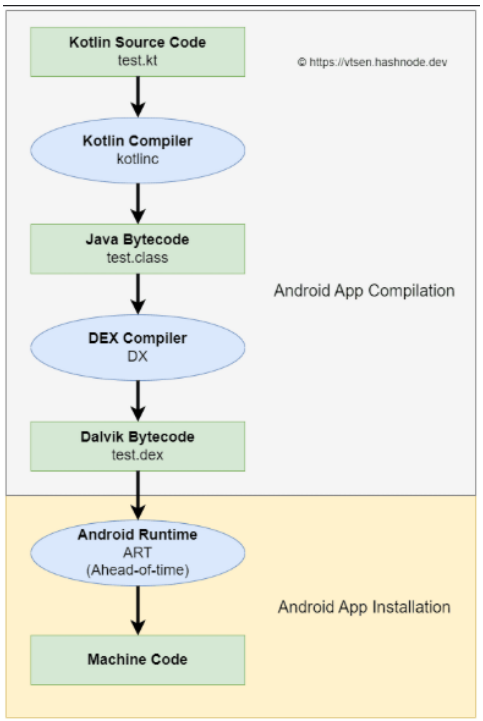


Рисунок 2.3 — Схема компіляції та встановлення Android-застосунку на Kotlin

## 2.2 Аналіз середовища для розробки мобільних додатків Android Studio

Android Studio є офіційним інтегрованим середовищем розробки (IDE) для створення мобільних застосунків на платформі Android. Воно підтримується компанією Google і ґрунтується на IntelliJ IDEA, розробленому компанією JetBrains. Android Studio забезпечує розробникам потужні інструменти для створення, тестування та розгортання мобільних застосунків, а також пропонує гнучку систему автоматизованого складання на основі Gradle, принцип роботи якого вкзанано на рис. 2.4 .[11] Середовище підтримує основні мови програмування, рекомендовані для Android-платформи, зокрема Kotlin — сучасну, безпечну та лаконічну мову, яка з 2019 року офіційно вважається пріоритетною для Android-розробки. Завдяки глибокій інтеграції з Android API, Kotlin дозволяє реалізовувати асинхронну обробку даних, безпечну роботу з null-значеннями та повну взаємодію з Java-бібліотеками.

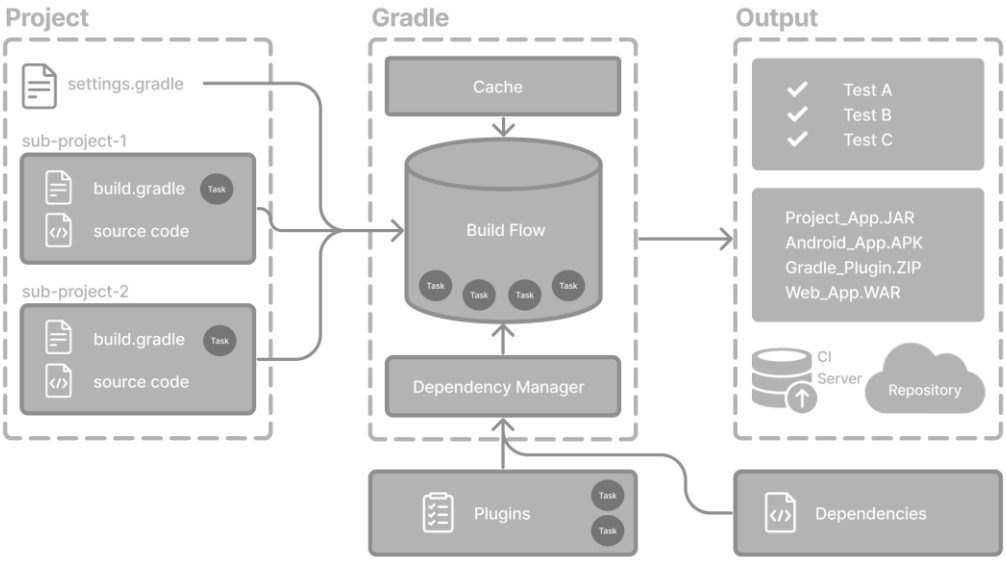


Рисунок 2.4 — Архітектура та процес складання проєкту в системі Gradle

Ключовими перевагами Android Studio є інтеграція з Android SDK і ADB (рис. 2.5) (Android Debug Bridge)[12], що дозволяє запускати застосунки на фізичних або віртуальних пристроях, аналізувати логування та профілювати продуктивність; підтримка Android Jetpack — набору бібліотек, що спрощують архітектуру застосунків (ViewModel, LiveData, Navigation, Room тощо); підтримка API CameraX — сучасного інтерфейсу для стабільної роботи з камерами на різних пристроях Android; використання системи збірки Gradle — для керування залежностями, CI/CD-процесами, конфігурацією складання; а також вбудований редактор інтерфейсів, що забезпечує створення адаптивних XML-розміток для різних типів екранів.

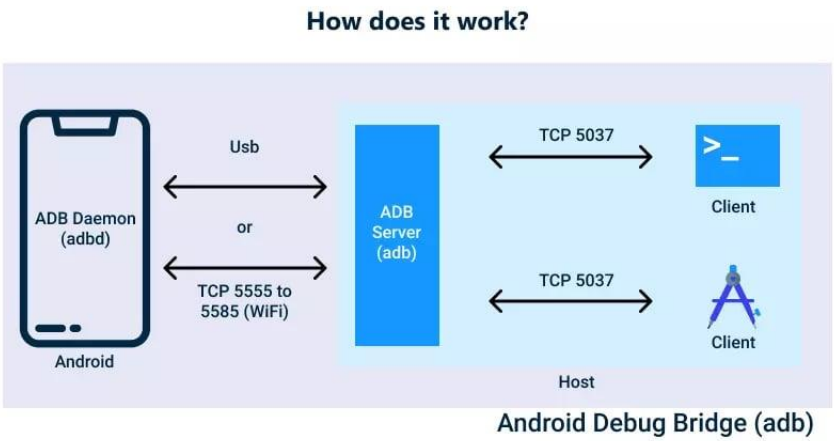


Рисунок 2.5 — Схема взаємодії компонентів Android Debug Bridge (ADB)

У межах реалізації інформаційної системи на базі Android Studio (рис. 2.6) було задіяно суміжні технології, які дозволили досягти високої ефективності. Зокрема, було реалізовано: обробку зображень у реальному часі — через захоплення кадрів із камери та їх подальшу підготовку (масштабування, нормалізація); інтеграцію моделей машинного навчання у форматі .tflite для локального виконання без підключення до хмарних сервісів; побудову вбудованого HTTP-сервера[13] для трансляції MJPEG-потоку відео, який можна переглядати в браузері; автоматичне формування PDF-звітів на основі збережених скріншотів; та логування подій — для збереження результатів інференції із часовими мітками.

Android Studio також має широку підтримку сторонніх бібліотек, необхідних для реалізації повноцінного функціоналу мобільного додатку[14]: MPAndroidChart і PDFBox — для графічної візуалізації та генерації документів; Kotlin Coroutines — для асинхронної обробки; OkHttp та NanoHTTPD — для HTTP-взаємодії; TensorFlow Lite та ONNX Runtime — для запуску моделей машинного навчання.

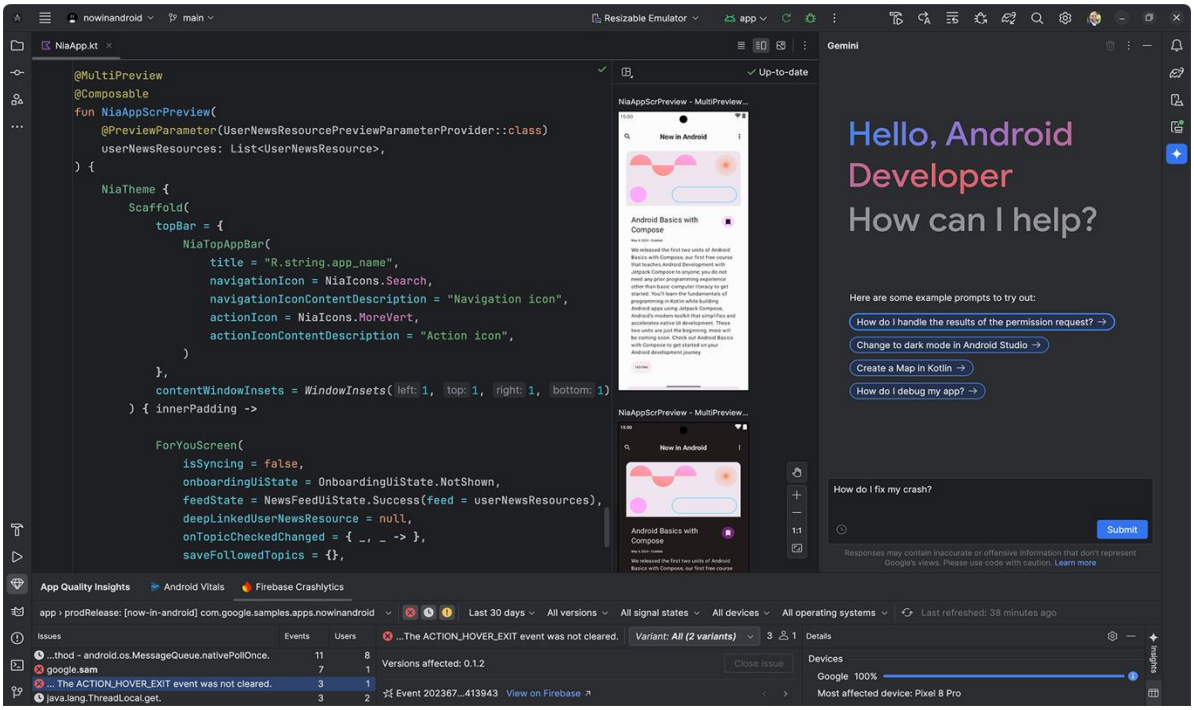


Рисунок 2.6 — Користувацький інтерфейс Android Studio

Таким чином, Android Studio виступає не лише як зручне середовище розробки, а як цілісна інфраструктурна платформа, що забезпечує всі необхідні технічні можливості для створення складного мобільного застосунку з інтеграцією нейромережевого аналізу, комп'ютерного зору, потокового відео та автоматизованого звітування.

## 2.3 Аналіз програмної платформи TensorFlow Lite

TensorFlow Lite — це платформа з відкритим кодом, створена Google для запуску моделей машинного навчання безпосередньо на мобільних пристроях, вбудованих системах та мікроконтролерах. Вона є полегшеною версією TensorFlow, адаптованою до роботи в умовах обмежених ресурсів, що забезпечує високу продуктивність при низькому енергоспоживанні [15].

Платформа надає інструменти для конвертації, оптимізації та виконання нейромережевих моделей локально, без підключення до мережі. Це особливо важливо для задач реального часу — зокрема комп'ютерного зору, обробки зображень і розпізнавання мови(рис. 2.7).

TensorFlow Lite має низку переваг, серед яких:

- підтримка квантованих і прунінг-моделей для зменшення ваги;
- виконання на CPU, GPU або прискорювачах (NNAPI, EdgeTPU);
- підтримка Android, iOS, Raspberry Pi;
- виклик моделей із Kotlin, Java, C++, ML Kit.

У межах проєкту TensorFlow Lite використовується як основа для локального запуску моделі виявлення об'єктів у відеопотоці. Це дозволяє уникнути передачі даних на сервер і забезпечує стабільну роботу навіть без інтернету.

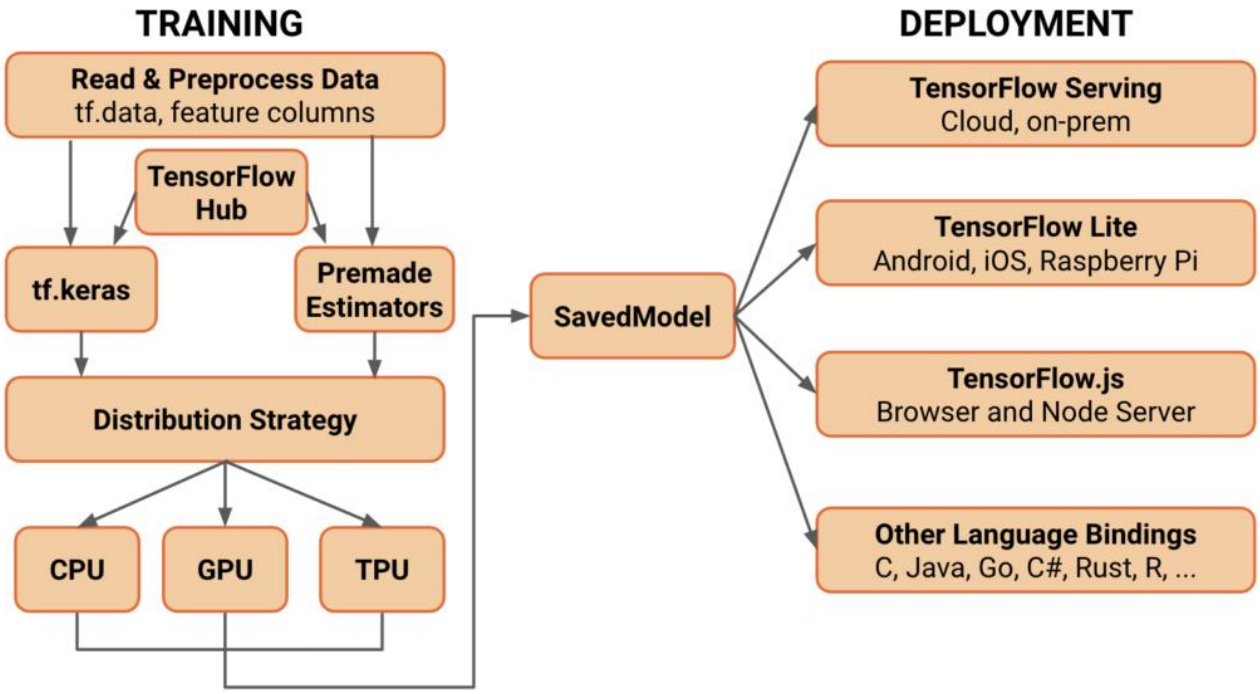


Рисунок 2.7 — Загальна схема навчання та розгортання моделей у TensorFlow

Серед ключових переваг платформи:

- продуктивність: інференція в реальному часі з мінімальними затримками;
- ефективність: економне використання ресурсів пристрою;
- конфіденційність: локальна обробка даних без зовнішніх сервісів;
- масштабованість: робота на широкому спектрі пристроїв;
- інтеграція: повна сумісність з Android Studio, підтримка GPU Delegate та NNAPI.

TensorFlow Lite успішно використовується в таких сценаріях, як розпізнавання об’єктів і облич у кадрі, класифікація аудіо та відео, виявлення аномалій або сигарет у мобільних системах моніторингу.

Таким чином, TFLite забезпечує необхідний баланс між точністю, швидкодією та автономністю, що робить його оптимальним вибором для реалізації нейромережевих детекторів на Android-платформі.

## 2.4 Вибір оптимальної моделі для виявлення об’єктів

Одним із ключових етапів створення нейромережевої інформаційної системи є вибір моделі глибинного навчання, здатної ефективно працювати в умовах обмежених ресурсів мобільного пристрою. Основними критеріями при виборі моделі стали: точність виявлення дрібних об’єктів, швидкодія на Android-платформі, можливість експорту у формат TensorFlow Lite та легкість інтеграції у мобільний застосунок.

У ході аналізу було розглянуто дві сучасні архітектури: YOLOv5 та YOLOv8[16] — легкі версії популярного сімейства моделей YOLO (You Only Look Once), призначених для роботи в реальному часі. Обидві моделі підтримуються бібліотекою ultralytics та можуть бути навчені на власному датасеті, однак мають суттєві архітектурні відмінності (рис. 2.8).

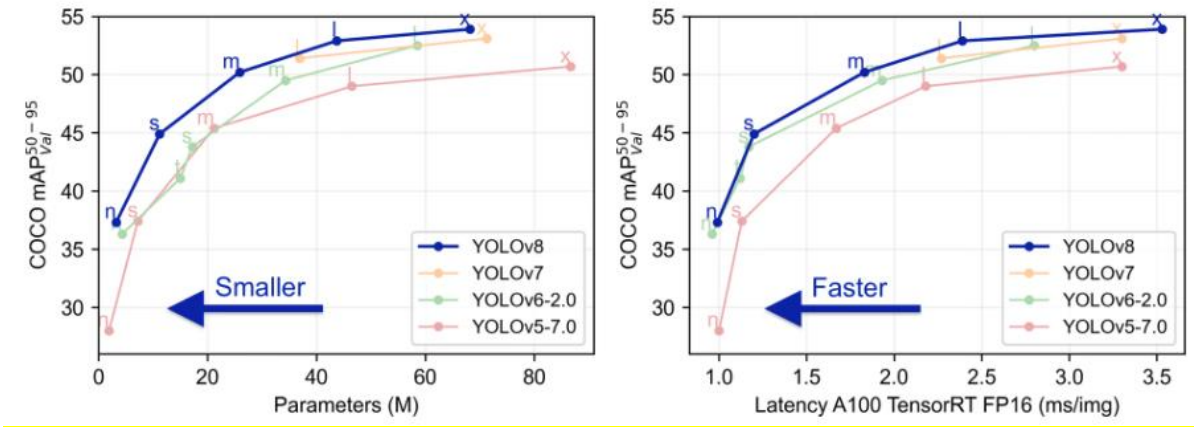


Рисунок 2.8 — Графік порівняння моделей YOLO

Модель YOLOv5[17], що з’явилася у 2020 році, використовує базову архітектуру CSPDarknet у поєднанні з PANet та anchor-based механізмом виявлення. Вона має стабільні показники точності та швидкості, однак менш оптимізована для розпізнавання дрібних об’єктів, таких як сигарети або pod-пристрої (рис. 2.9).

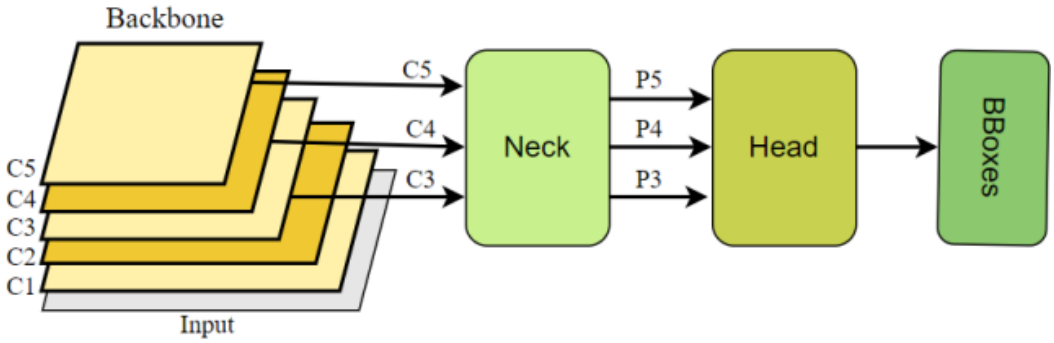


Рисунок 2.9 — Структура моделі YOLOv5

Крім того, її структура менш гнучка при адаптації до мобільних форматів.

Результати для першого запуску (рис. 2.10):

– mAP@0.5 = 0.716

– mAP@0.5:0.95 = 0.486

– Precision = 0.892

– Recall = 0.613

Основні метрики оцінювання якості моделі:

– mAP@0.5 (Mean Average Precision при IoU ≥ 0.5) — відображає середню точність виявлення об'єктів при помірному порозі перекриття. Значення в межах 0.7–0.9 вважаються хорошими. Формально mAP визначається як:

$$mAP = \frac{1}{N}\sum(\text{Precision на кожному рівні Recall})$$

– mAP@0.5:0.95 — суворіша метрика, що обчислюється як середнє значення mAP для кількох рівнів перекриття (від 0.5 до 0.95 з кроком 0.05). Цей показник дозволяє оцінити, наскільки модель є стабільною при різних рівнях точності позиціонування.

– Precision (точність) — частка коректно виявлених об'єктів серед усіх позитивних передбачень [18]. Високе значення precision свідчить про низький рівень хибних спрацювань та визначається за формулою:

$$Precision = \frac{TP}{TP + FP}$$

де $TP$ — кількість правильних позитивних спрацювань, $FP$ — кількість хибнопозитивних.

– Recall (повнота) — частка правильно виявлених об'єктів серед усіх об'єктів на зображеннях. Високе значення recall означає, що модель рідко пропускає об'єкти, вираховується за формулою:

$$Recall = \frac{TP}{TP + FN}$$

де $FN$ — кількість об’єктів, які модель пропустила.

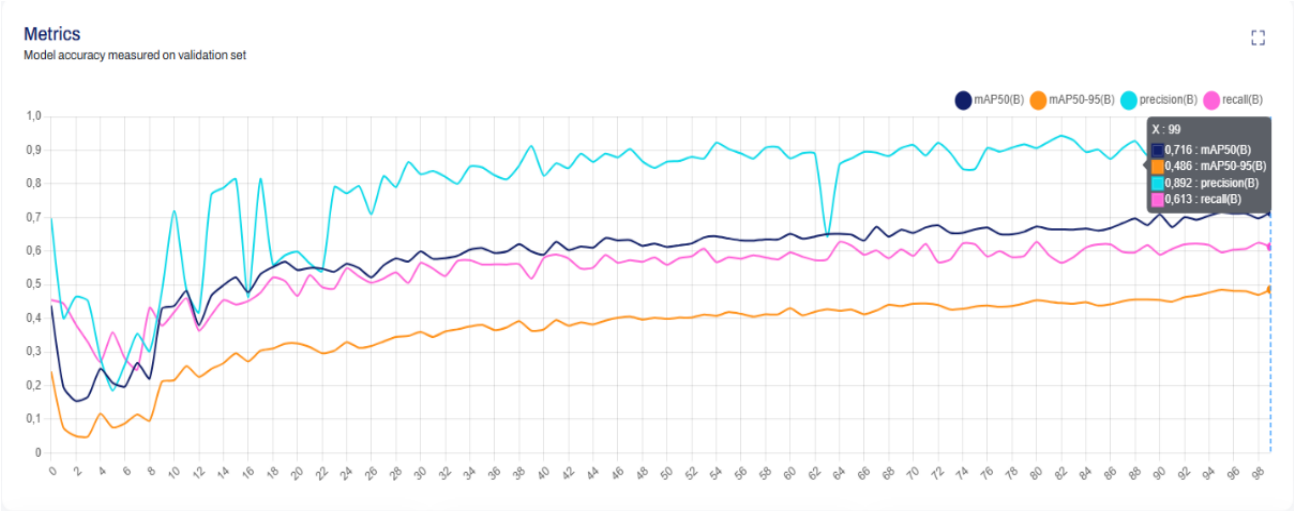


Рисунок 2.10 — Метрики моделі YOLOv5

Натомість YOLOv8s, представлена у 2023 році, є сучаснішою й компактнішою моделлю, що використовує anchor-free підхід, покращену neck-архітектуру C2f та вдосконалену обробку ознак (рис. 2.11). Вона демонструє вищу якість виявлення, особливо для невеликих об’єктів, та забезпечує кращу продуктивність при інференції на мобільних пристроях. Завдяки спрощеній структурі (рис. 2.12) модель легко експортується у формати ONNX та TFLite, що значно полегшує інтеграцію у застосунок на базі Android.

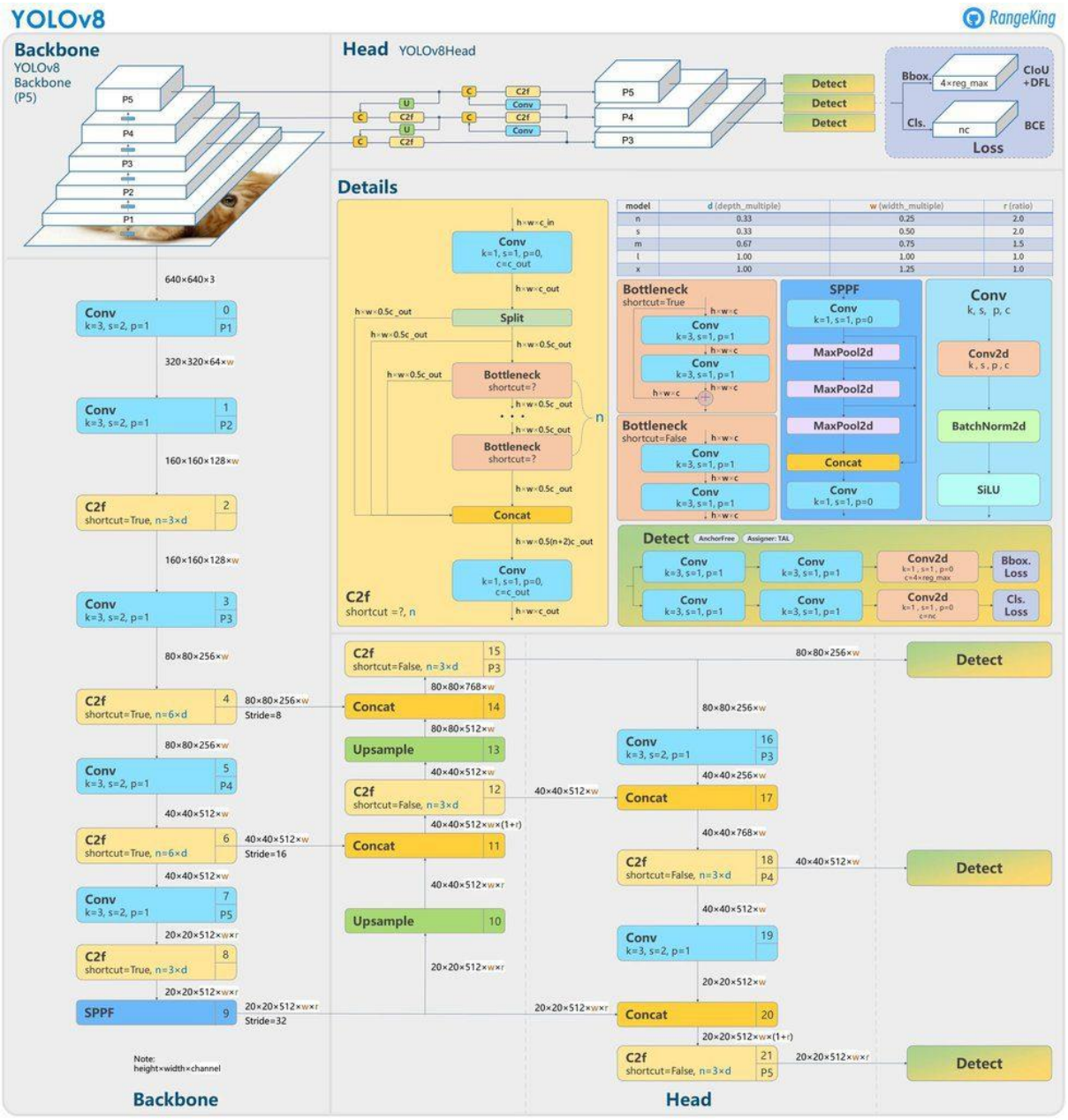


Рисунок 2.11 — Архітектура моделі YOLOv8

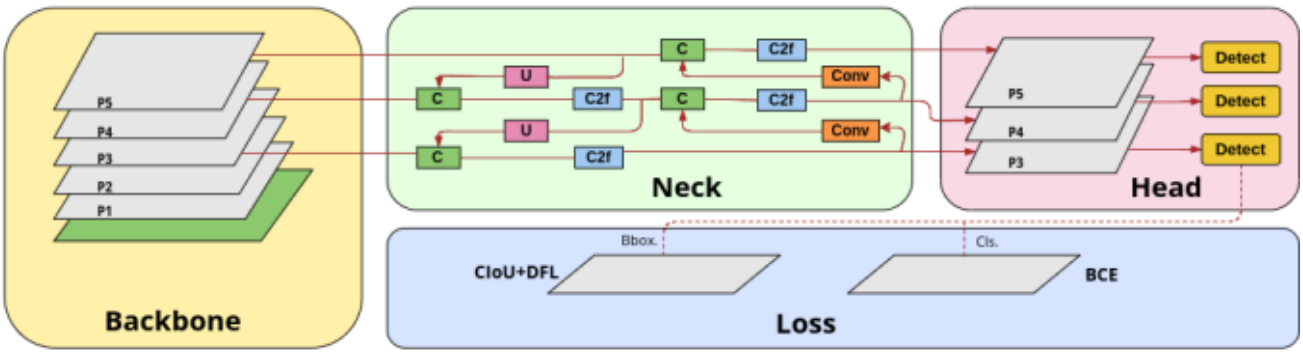


Рисунок 2.12 — Структура моделі YOLOv8

Результати для другого запуску (рис. 2.13):

– mAP@0.5 = 0.904
– mAP@0.5:0.95 = 0.620
– Precision = 0.877
– Recall = 0.820

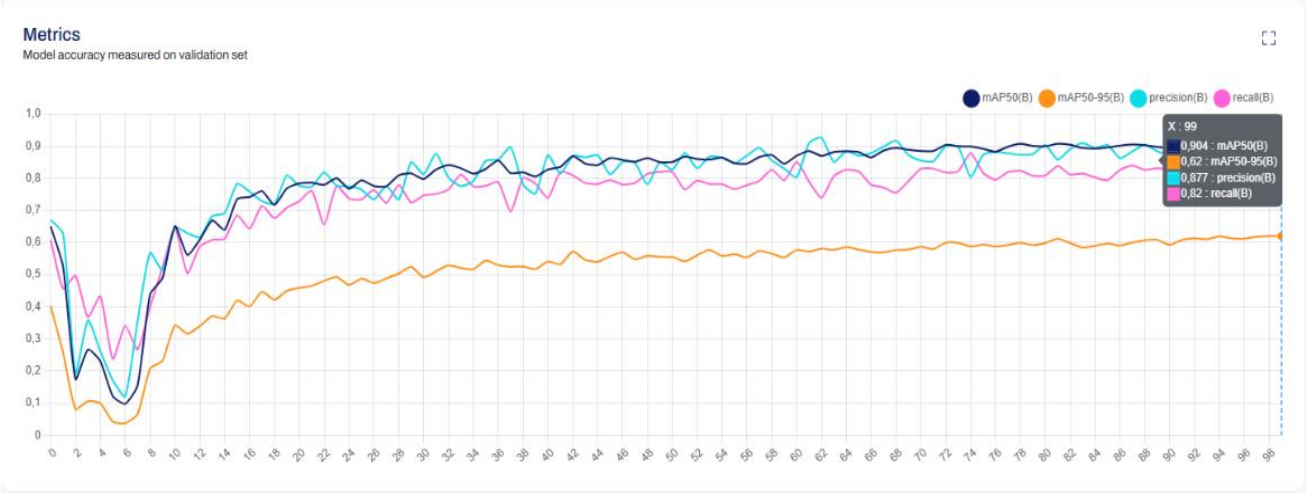


Рисунок 2.13 — Метрики моделі YOLOv8s

Аналіз наведених метрик свідчить, що модель другого запуску показала значно кращу узагальнюючу здатність (на що вказує зростання mAP@0.5:0.95), а також помітно вищу повноту (recall = 0.820 проти 0.613), що є критично важливим для задачі виявлення куріння, де необхідно мінімізувати пропуск об'єктів. Значення точності при цьому залишилося на високому рівні в обох випадках.

Таким чином, модель другого запуску демонструє кращий баланс між точністю і повнотою, а також стабільнішу роботу за різних умов, що робить її більш придатною для практичного застосування в рамках інформаційної системи виявлення курців (рис. 2.14).

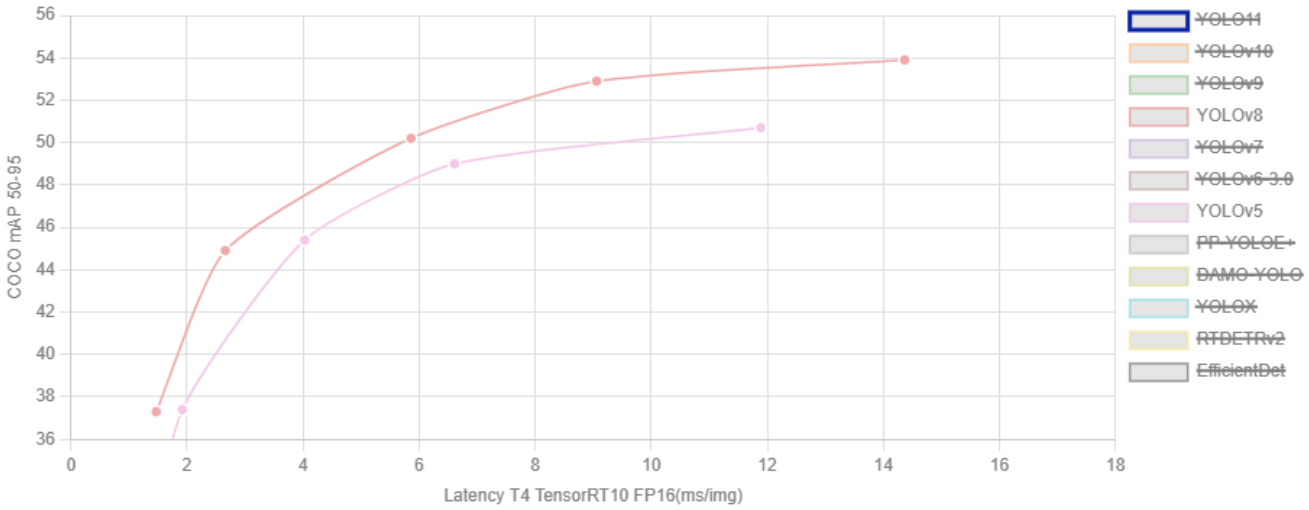


Рисунок 2.14 — Порівняльний графік ефективності моделей YOLOv8 та YOLOv5

Перед початком фінального навчання було протестовано різні інструменти. Спершу використовувалась платформа Roboflow[19] для завантаження й анотації початкового набору зображень (811 зразків), а також для проведення базових експериментів. Згодом, з метою поліпшення гнучкості керування моделлю, було здійснено перехід на платформу Ultralytics HUB[20], яка забезпечила зручний інтерфейс для тренування YOLOv8s та підготовки моделі до експорту.

Модель була навчена у середовищі Kaggle[21] із використанням GPU, після чого експортована у формат .tflite та вбудована у мобільний застосунок. Такий підхід дозволив реалізувати повністю автономну систему виявлення курців на мобільному пристрої, здатну працювати в реальному часі без підключення до мережі Інтернет.

## 2.5 Висновки

У цьому розділі було обґрунтовано вибір оптимальних інформаційних технологій для реалізації нейромережевої системи виявлення куріння на базі мобільного пристрою. Зокрема, доведено доцільність використання мови програмування Kotlin як основного інструменту розробки Android-застосунку,

Android Studio як інтегрованого середовища розробки, а також платформи TensorFlow Lite для локального виконання моделей машинного навчання.

Окрему увагу приділено вибору архітектури моделі глибинного навчання. У результаті аналізу обрано YOLOv8 — сучасну, легку та продуктивну модель, оптимізовану для виявлення дрібних об’єктів, яка ефективно працює на мобільних пристроях. Для тренування та порівняння моделей було використано платформи Roboflow та Ultralytics HUB, що дозволило створити повноцінну інфраструктуру розробки і порівняння нейромережевих моделей.

Застосування вищезазначених технологій у розглянутих рішеннях демонструє можливість побудови стабільних систем реального часу, здатних автономно виявляти порушення навіть за відсутності підключення до мережі Інтернет. Такі системи є масштабованими та можуть бути адаптовані або доповнені новим функціоналом у майбутніх версіях.

# 3 ОПИС ПРОЦЕСУ РОЗРОБЛЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

## 3.1 Формування набору даних для тренування моделі

Для успішного навчання нейронної мережі, здатної виявляти факти куріння у відеопотоці, необхідним етапом є формування якісного та збалансованого набору даних (dataset) [22]. Саме від повноти, різноманітності та точності анотацій залежить ефективність моделі та рівень її узагальнення на нових прикладах.

Початковий датасет, який використовувався на етапі експериментів, складався з 811 об'єктів на 465 навчальних та 125 валідаційних зображеннях і охоплював два класи: Person (людина) та Cigarette (сигарета). Проте виявилось, що для досягнення високої точності цього обсягу недостатньо, особливо в умовах реального відеопотоку.

З метою покращення результатів модельного навчання датасет було суттєво розширено. Нові зображення були зібрані вручну з різних відкритих джерел — Google Images, Reddit, YouTube (кадри з відео), спеціалізованих форумів, а також із реальних зйомок, зроблених камерою мобільного пристрою. Усі зображення були вручну анотовані, тобто кожен об’єкт на кадрі був обведений і класифікований у відповідний клас (рис. 3.1).

У фінальній версії датасету було виділено три класи для детекції:

- Person — людина;
- Cigarette — сигарета;
- Vape — електронна сигарета або pod-система.

Для розмітки використовувалися сервіс Roboflow, які дозволяють швидко створювати анотації у форматі YOLO. Дані були експортовані у формат, сумісний із фреймворком Ultralytics YOLOv8, зі структурою: images/train, images/val, images/test, labels/train, labels/val, labels/test (рис. 3.2).

Загальний обсяг оновленого датасету склав:

- 2496 зображень загалом;
- 1802 зображень у навчальній вибірці (72%);

- 497 зображень у валідаційній вибірці (20%);
- 197 зображень у тестовій вибірці (8%).

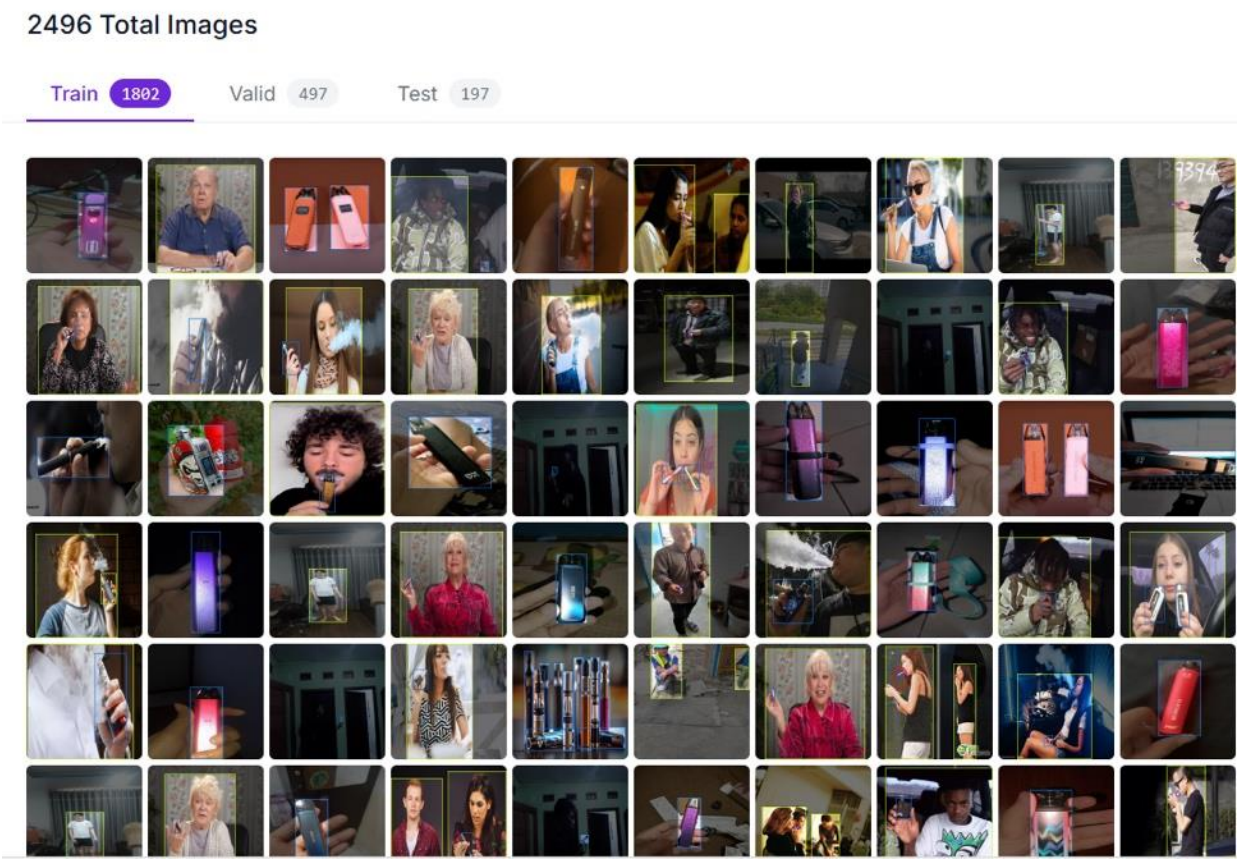


Рисунок 3.1 — Фрагмент розміченого датасету

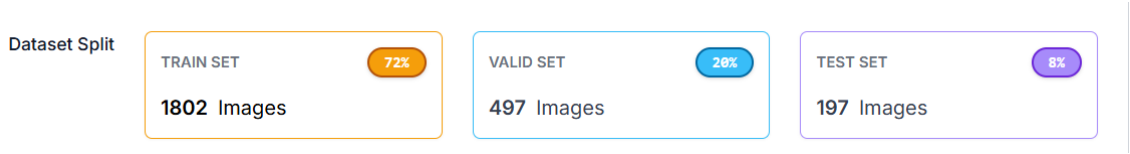


Рисунок 3.2 — Розподіл підмножин: train / val / test

Ретельна ручна розмітка та поступове нарощування обсягу даних дозволили забезпечити високу якість подальшого навчання моделі та покращили її стійкість до типових сценаріїв споживання тютюнових виробів.

3.2 Розроблення мобільного додатку та подальше впровадження моделі виявлення об’єктів

На етапі реалізації програмного продукту було розроблено мобільний Android-застосунок, який забезпечує автоматичне виявлення фактів куріння на основі потокового відео з камери пристрою. Основною метою стала розробка рішення, яке поєднує зручний інтерфейс користувача з ефективною роботою нейронної моделі без потреби у серверних обчисленнях.

Застосунок реалізовано мовою Kotlin у середовищі Android Studio, із використанням таких ключових компонентів:

- CameraX API — захоплення відеопотоку з камери пристрою;
- ImageAnalysis — передача кадрів у нейронну модель у режимі реального часу;
- TensorFlow Lite Interpreter — виконання нейронної моделі безпосередньо на пристрої;
- MJPEG-сервер (на базі NanoHTTPD) — трансляція обробленого відеопотоку через браузер у локальній мережі;
- Canvas API — накладання результатів детекції у вигляді рамок та текстових міток.

З метою зручності користувача було реалізовано веб-інтерфейс, що дозволяє користувачеві в режимі реального часу переглядати відео-трансляцію з мережевих камер. Завдяки адаптивному дизайну, він доступний з будь-яких пристроїв — комп’ютерів, планшетів чи смартфонів, підключених до тієї самої локальної мережі. Це забезпечує максимальну зручність: достатньо відкрити браузер та перейти за вказаним URL, і користувач вже може моніторити обстановку без необхідності встановлення додаткових програм[23].

Інтерфейс оснащений функцією автоматичної детекції подій: він аналізує відеопотік і підсвічує кадри, де виявлено куріння. Крім того, користувач може вручну перемикати камери у випадку, якщо система підтримує декілька відеоджерел. Це дозволяє оперативно реагувати на зміни ситуації та переключатися між різними зонами спостереження.

Для фіксації важливих моментів реалізовано можливість збереження скріншотів одним кліком. В подальшому користувач може сформувати структурований PDF-звіт, що містить час, зображення подій та технічні

коментарі. Це значно полегшує процес використання застосунку (рис. 3.3) та документування інцидентів і підвищує рівень контролю та безпеки об’єкта або приміщення.

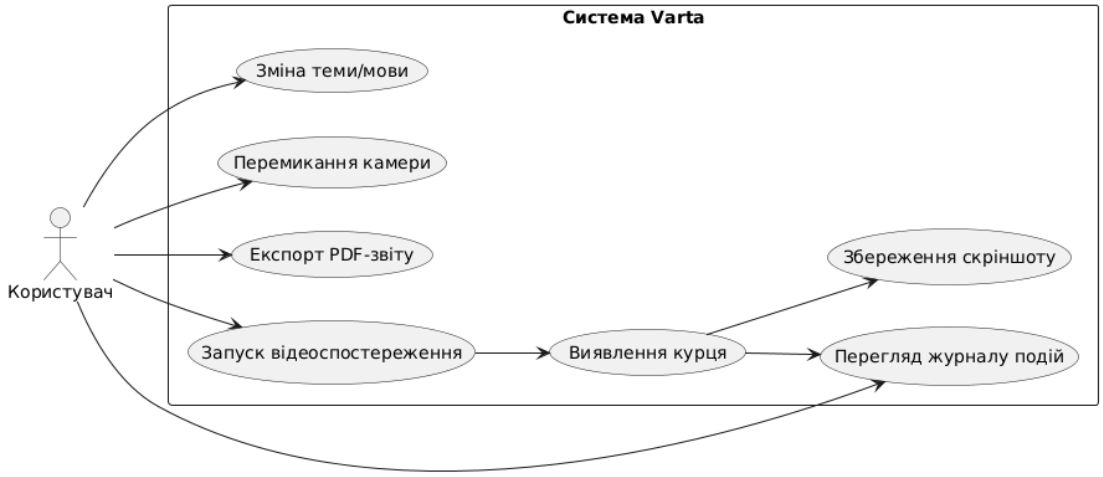


Рисунок 3.3 — Діаграма використання мобільного застосунку

3.3 Аналіз якості тренування нейромережевої моделі

Початково проєкт використовував модель YOLOv8s, однак із розширенням датасету до 2496 зображень та застосуваннням різноманітних прийомів аугментацій (віддзеркалення, зміна яскравості, повороти, накладання шуму), кількість зображень зросла з 2496 до понад 6100 (рис. 3.4), розділено на відповідні підмножини(рис. 3.5), а також було включено третій клас (vape-pod), що суттєво підняло вимоги до швидкодії. Тому модель було перелаштовано на легшу версію YOLOv8n, яка має меншу кількість параметрів і краще підходить для мобільних пристроїв, забезпечуючи стабільну продуктивність[24].

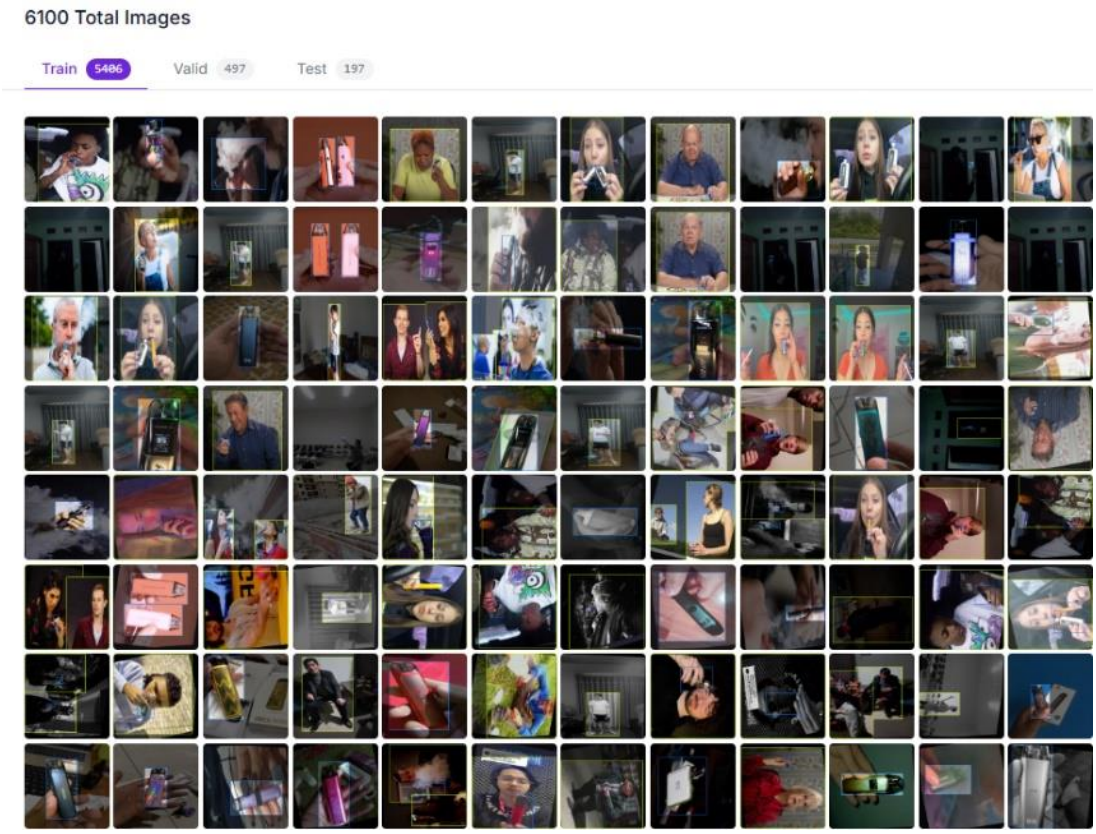


Рисунок 3.4 — Фрагмент фінального датасету

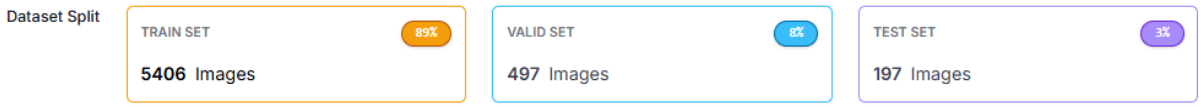


Рисунок 3.5 — Фінальний розподіл підмножин: train / val / test

Модель навчалась протягом 100 епох із розміром зображення 640×640, batch size = 16 (Кількість зображень, що обробляються за один крок навчання) та використанням попередньо натренованих ваг. Подано графік зміни функцій втрат(loss) та основних метрик якості впродовж 100 епох навчання моделі.[25] Кожен графік включає фактичні значення (results) та згладжену криву (smooth) для зручнішого сприйняття динаміки (рис. 3.6).

Метрики точності:

– Precision (metrics/precision(B)) зросла з ~0.4 до ~0.89, що означає суттєве зменшення кількості хибних спрацьовувань.

– Recall (metrics/recall(B)) досягла ~0.82, що вказує на високу здатність моделі виявляти всі об’єкти класів.

– mAP@0.5 поступово зросла до ~0.90, що є відмінним результатом для задачі об'єктного виявлення.

– mAP@0.5:0.95 досягла ~0.62, демонструючи стабільну якість моделі навіть при більш жорстких порогах перекриття (IoU)[25].

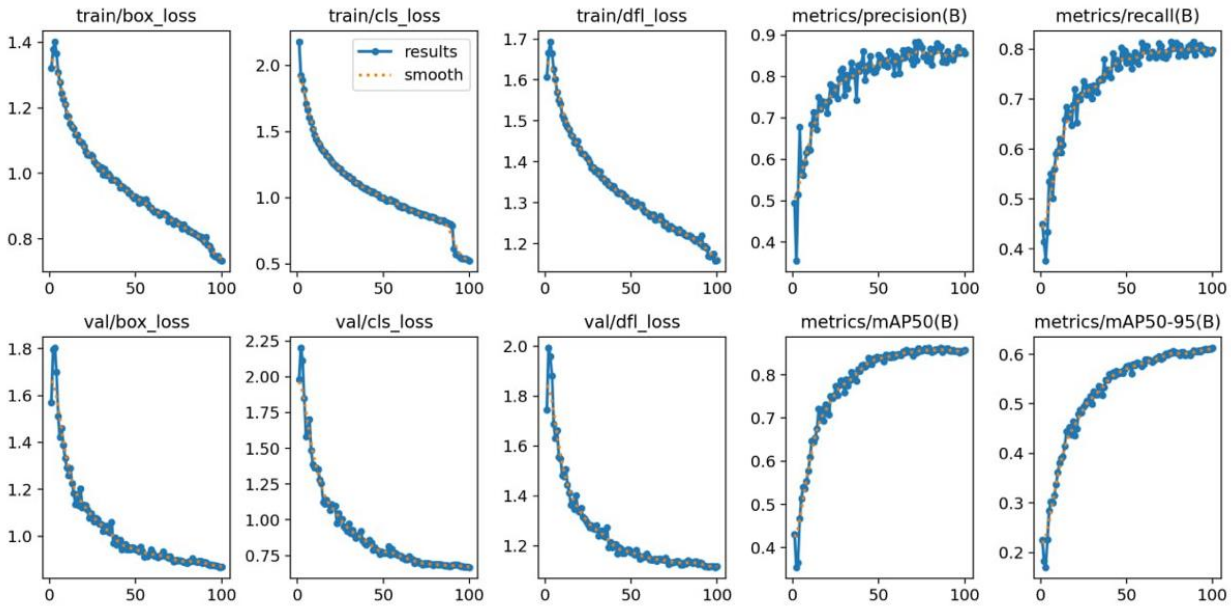


Рисунок 3.6 – Графіки train/val втрат (box_loss, cls_loss, dfl_loss) та основних метрик

Також подано нормалізовану матрицю помилок (рис. 3.7), яка відображає точність класифікації кожного класу: Cigarette, Person, vape-pod та background. Значення в матриці подані у відсотковому співвідношенні (від 0 до 1), що дозволяє оцінити частоту коректних і хибних передбачень моделі по кожному класу.

Ключові спостереження:

– Клас Person демонструє найвищу точність класифікації: 92% зразків було ідентифіковано правильно. Це свідчить про чітке виявлення людей на зображеннях.

– Клас Cigarette класифікується з точністю 78%, але близько 17% таких об’єктів модель помилково відносить до background, що вказує на виклики у виявленні сигарет при слабкій видимості або шумному фоні.

– Клас vape-pod має помітне перекриття з background (58%) та Cigarette (88%), що є основним джерелом помилок. Це, ймовірно, пов’язано зі схожістю візуальних ознак між електронними сигаретами та звичайними.

– Клас background частково перетинається з іншими класами, зокрема — помилково класифікує 21% сигарет як фон, що підтверджує складність задачі у випадках часткового перекриття або малого розміру об’єкта.

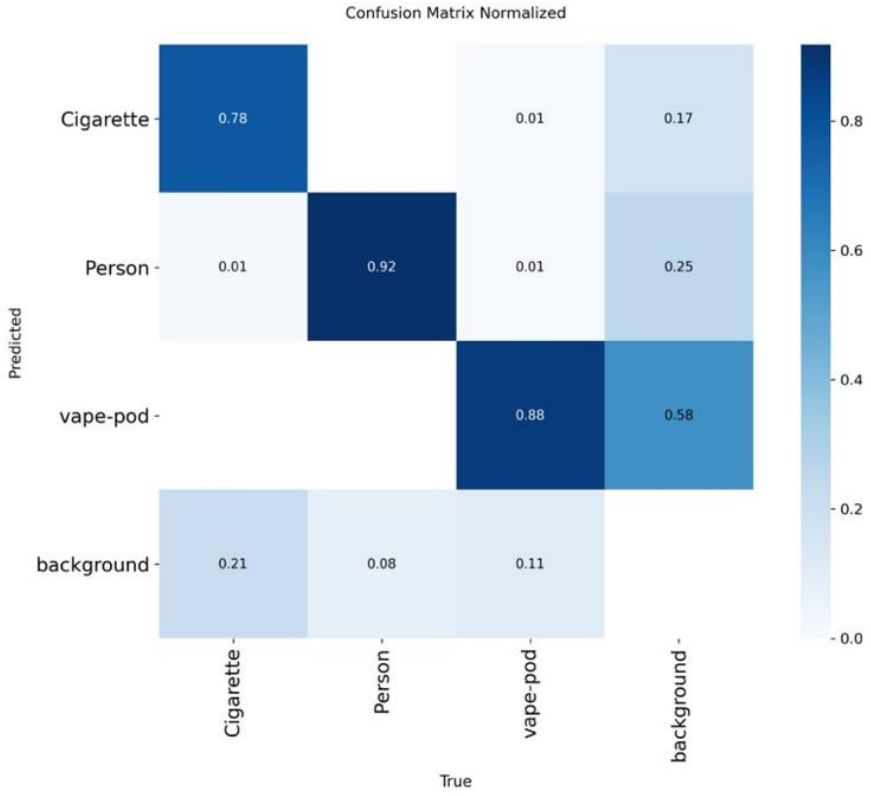


Рисунок 3.7 – Нормалізована матриця помилок моделі YOLOv8n після 100 епох навчання

Представлено абсолютну матрицю помилок (confusion matrix), яка містить абсолютну кількість правильних та хибних передбачень моделі для кожного з чотирьох класів: Cigarette, Person, vape-pod, background (рис. 3.7).

Основні спостереження:

– Клас Person: з 442 зразків модель правильно класифікувала 382 (86%), помилилася у 56 випадках, відносячи об'єкти до фону.

– Клас Cigarette: 173 зразки передбачено правильно, 38 — хибно віднесено до фону, а ще 4 — переплутано з Person або vape-pod.

– Клас vape-pod: 309 зразків правильно, тоді як 130 випадків (≈30%) модель класифікувала як background, що підтверджує складність візуального розмежування цього класу.

– Клас background: тут виникає найбільша кількість хибнопозитивних передбачень — загалом 118 об'єктів (Cigarette, Person, vape-pod) було віднесено до фону, з них 47 — це сигарети.

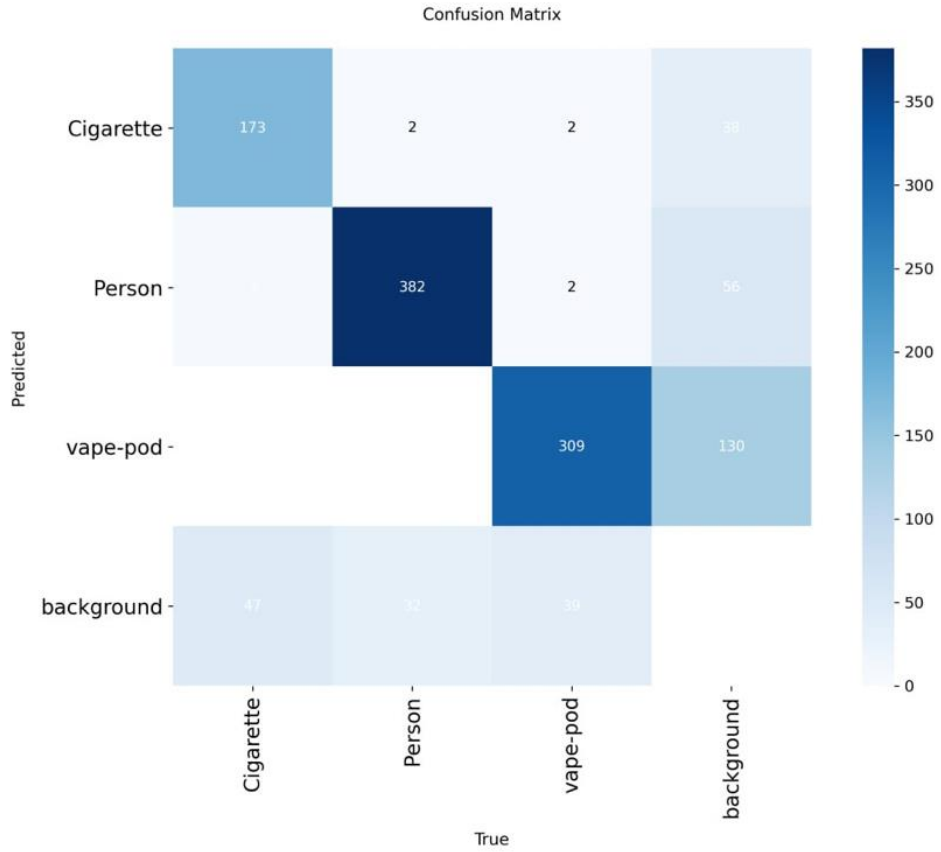


Рис. 3.8 – Абсолютна матриця помилок (кількість правильних та хибних передбачень за класами)

Ще представлено графік залежності Recall від порогу впевненості (Confidence) для кожного з трьох класів (Cigarette, Person, vape-pod) та усереднене значення для всіх класів (рис. 3.9).

Інтерпретація графіка:

– Клас Person (помаранчева лінія) демонструє найвищу стабільність — повнота зберігається на рівні понад 80% навіть при високих значеннях confidence > 0.8. Це свідчить про високий рівень впевненості моделі при виявленні людей.

– Клас vape-pod (зелена лінія) також показує доволі стабільну поведінку, хоча зі зниженням до ~70% recall при порогах вище 0.85.

– Клас Cigarette (блакитна лінія) має найнижчу стійкість до підвищення порогу: при confidence > 0.7 recall різко падає, що означає, що модель менш впевнена у виявленні сигарет.

– Загальна лінія (синя) показує середній показник по всіх класах — максимальний recall становить 0.94 при нульовому порозі впевненості.

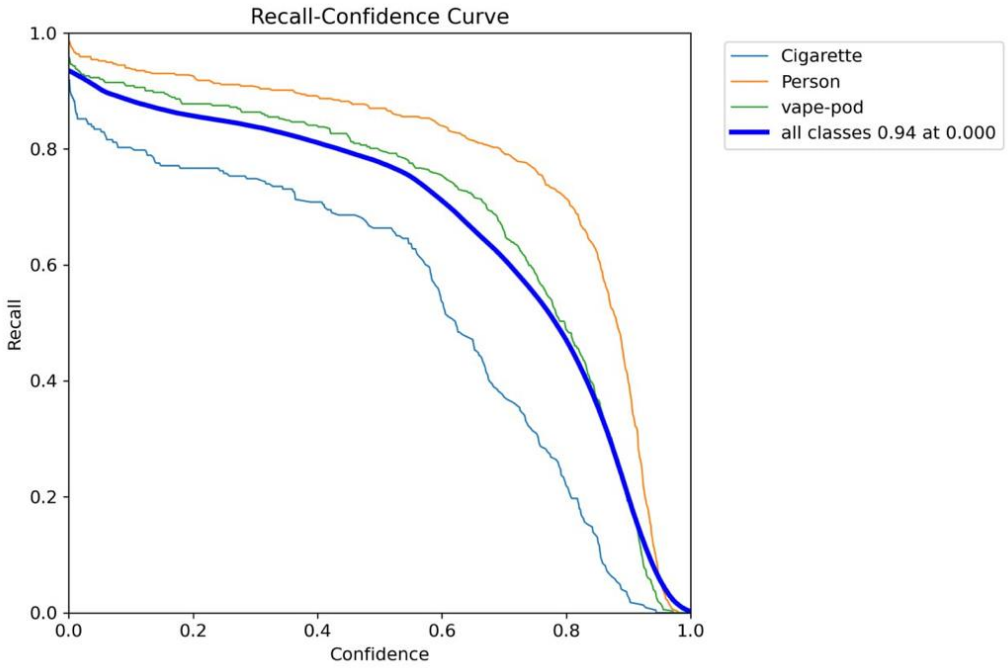


Рис. 3.9 – Крива залежності Recall від Confidence для кожного класу та усередненого значення

Отримано графік залежності Precision від порогу впевненості (Confidence). Він демонструє, наскільки зростає точність передбачень моделі при підвищенні порогу confidence, тобто, наскільки впевнено модель виявляє об’єкти без помилкових спрацьовувань (рис. 3.10).

Інтерпретація результатів:

– Клас Person (помаранчева лінія) зберігає високу точність протягом усього діапазону confidence. Починаючи вже з 0.3 precision перевищує 80%, а при порогах понад 0.8 — майже досягає 1.0. Це підтверджує, що виявлення людей є найбільш надійним.

– Клас vape-pod (зелена лінія) демонструє поступове зростання точності до ~0.95, але більш повільно, що свідчить про більшу кількість хибнопозитивних детекцій при низькому порозі.

– Клас Cigarette (блакитна лінія) також досягає високої точності, але з великим розкидом у низькому діапазоні (до 0.6), що вказує на складність і варіативність цього класу в датасеті.

– Усереднена лінія (синя) досягає значення 0.99 при confidence = 1.0, що підтверджує здатність моделі забезпечити практично ідеальну точність за умови високого порогу.

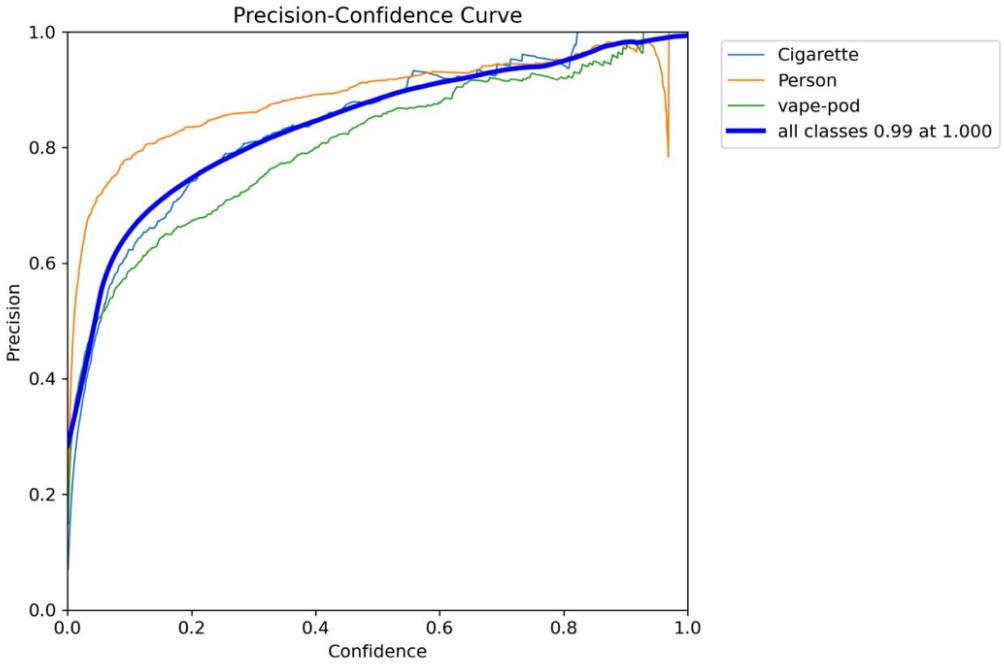


Рис. 3.10 – Крива залежності Precision від Confidence для кожного класу та усередненого значення

Зображено залежність F1-метрики від порогу впевненості (Confidence) для кожного класу (рис 3.11). F1-score є гармонійним середнім між Precision та Recall, що дозволяє оцінити загальну якість класифікації при різних порогах.

Ключові спостереження:

– Клас Person (помаранчева лінія) демонструє найкращий F1-score, що стабільно утримується вище 0.85 у широкому діапазоні порогів (від ~0.2 до ~0.8). Це підтверджує надійність моделі у виявленні людей.

– Клас vape-pod (зелена лінія) досягає пікового F1 близько 0.82 при порозі ~0.4, але швидко втрачає якість при збільшенні порогу.

– Клас Cigarette (блакитна лінія) показує найменшу стабільність: максимальний F1 ~0.77 досягається при порозі ~0.35–0.4, але зростання порогу швидко погіршує якість класифікації.

– Сумарна F1-крива (синя лінія) досягає максимуму 0.83 при порозі 0.438, що можна вважати оптимальним значенням confidence threshold для загальної системи.

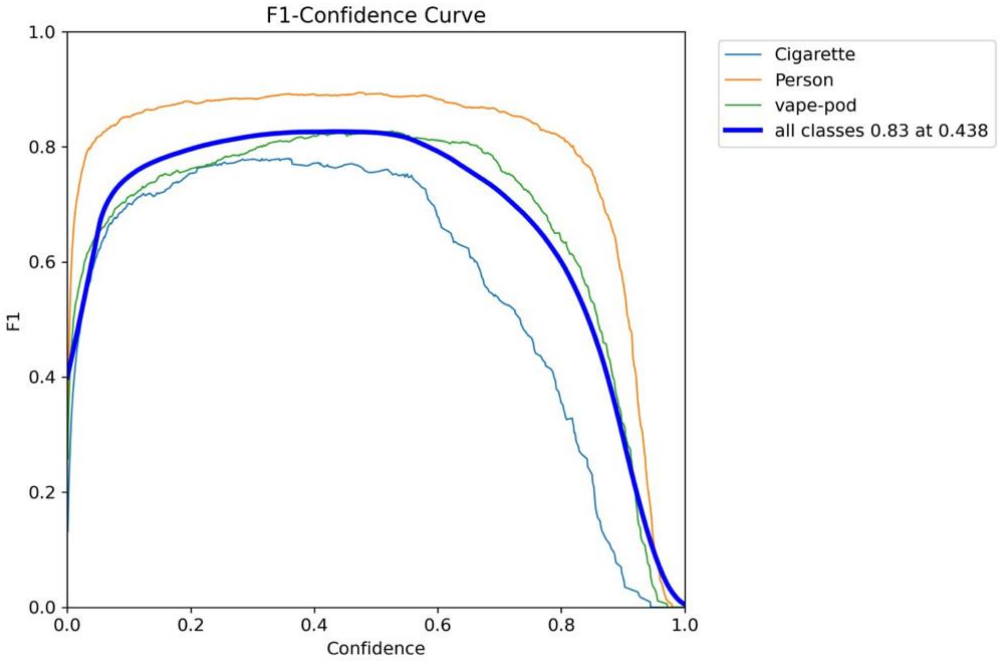


Рис. 3.11 – Залежність F1-метрики від порогу впевненості для кожного класу та середнього значення

## 3.4 Опис IT-інфраструктури та архітектури інформаційної системи

Інформаційна система, реалізована у межах даного проєкту, побудована як автономний мобільний програмний комплекс, що поєднує локальну обробку відеопотоку із можливістю взаємодії через вбудований веб-інтерфейс. Архітектура системи є модульною, що дозволяє легко масштабувати функціональність та адаптувати її до нових сценаріїв використання, зберігаючи при цьому автономність та конфіденційність.

Основні компоненти системи:

- Клієнтський пристрій (Android-смартфон) — центральний елемент системи. Він виконує захоплення відео, нейромережеву інференцію, збереження результатів та трансляцію MJPEG-потоку. На пристрої розгорнуто мобільний застосунок, що реалізує взаємодію з камерою, обробку зображення за допомогою TFLite-моделі YOLOv8 та забезпечує інтеграцію з веб-інтерфейсом.
- Вбудований локальний веб-сервер (MJPEG-сервер) — реалізований за допомогою бібліотеки NanoHTTPD. Забезпечує трансляцію відео у форматі MJPEG, доступну з браузера за адресою типу http://<IP>:8080. Сервер обробляє маршрути , /stream, /motion-status, /switch-camera тощо.
- Браузер користувача — забезпечує віддалений доступ до інтерфейсу застосунку через HTML-сторінку. Реалізовано JavaScript-логіку для перегляду відео, перемикання камер, перегляду збережених кадрів та генерації PDF-звітів. Сторінка включає динамічний Canvas API для візуалізації об’єктів детекції (bounding boxes).
- Локальне файлове сховище — використовується для збереження скріншотів із детекцією, логів подій із часовими мітками, а також підготовки матеріалів для формування PDF-звітів. Уся інформація зберігається локально на пристрої, що гарантує конфіденційність.

На UML-діаграмі ілюстровано структуру основних класів системи: MainActivity, TFLiteModelHelper, MJPEGServer, ImageAnalyzer, їхню взаємодію та відповідальність за різні функціональні блоки. Завдяки чіткому поділу обов’язків забезпечується гнучкість, розширюваність та простота супроводу коду (рис. 3.12).

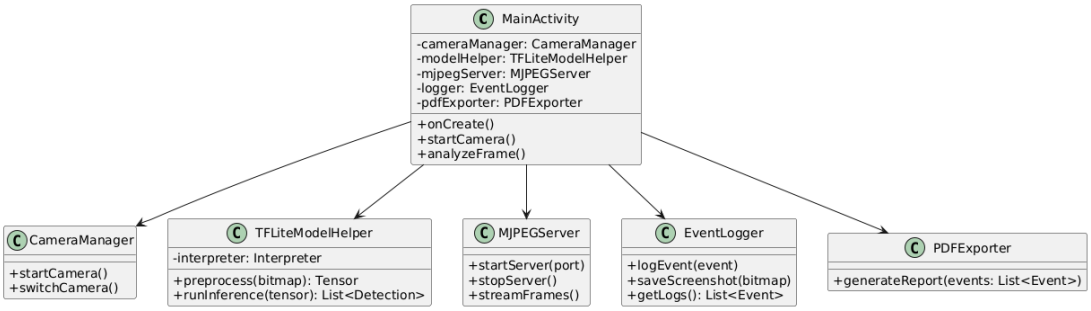


Рисунок 3.12 — Діаграма класів мобільного застосунку.

Архітектурні особливості

Загальна архітектура є однорівневою (single-tier): усі обчислення та сервіси працюють локально на одному пристрої (рис. 3.13).

Такий підхід забезпечує:

- повну автономність;
- відсутність потреби у зовнішньому сервері;
- збереження приватності;
- низьку затримку при обробці кадрів;
- стабільну продуктивність на пристроях середнього рівня.

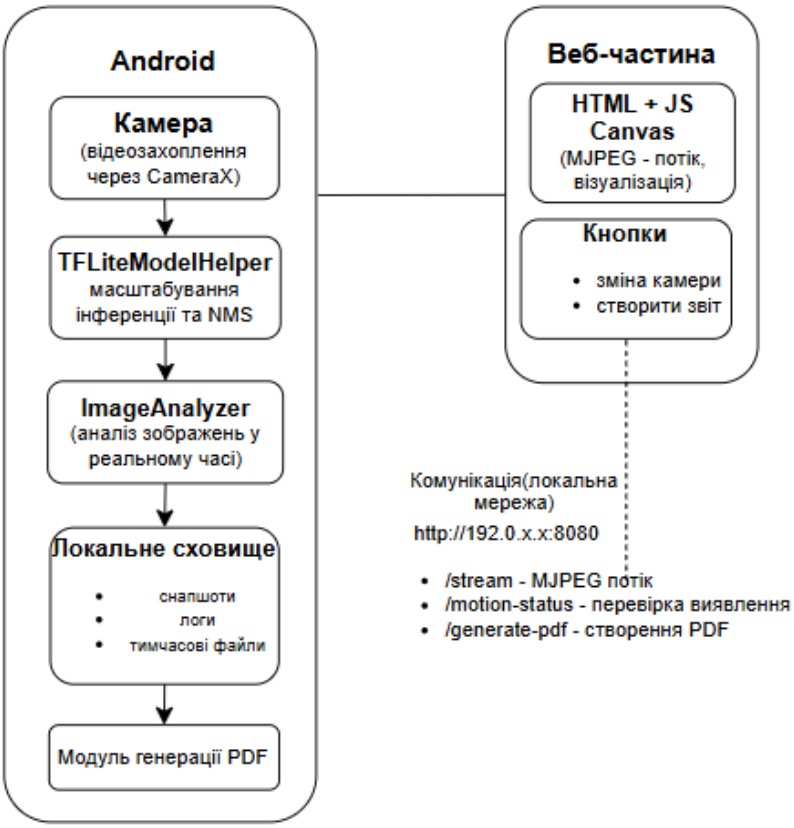


Рисунок 3.13 — Архітектурна діаграма інформаційної системи.

На схемі (рис. 3.14) представлено логічну взаємодію між компонентами: камера → інференція → локальне збереження результатів → трансляція MJPEG → HTML-інтерфейс → взаємодія з користувачем.

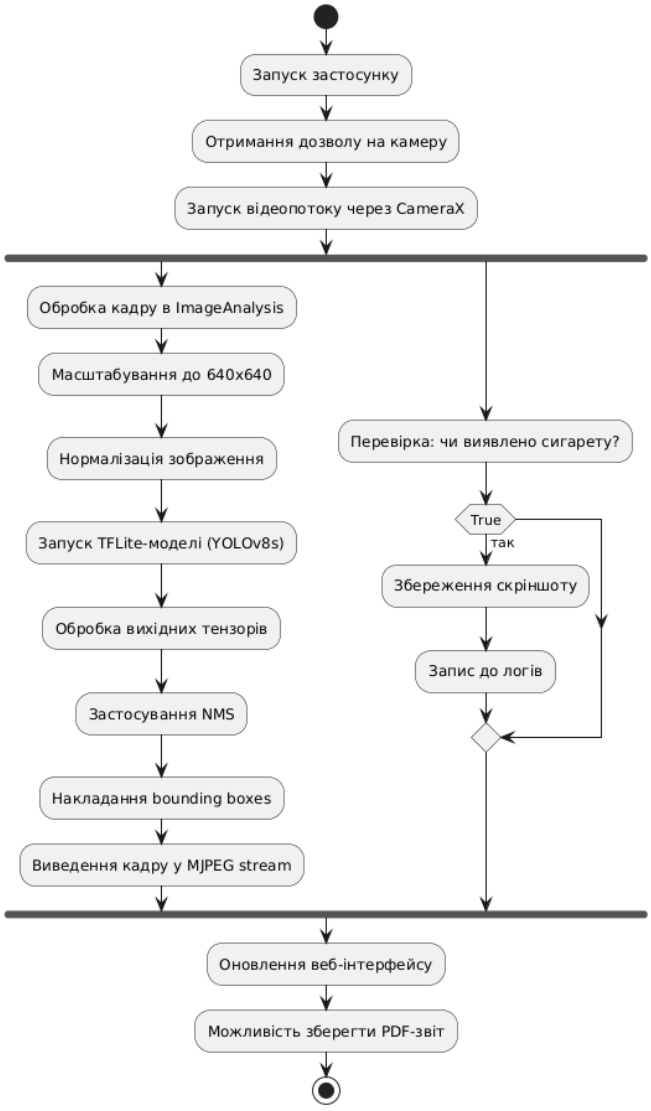


Рисунок 3.14 — Діаграма активності роботи мобільного застосунку

Процес розгортання моделі та застосунку складається з наступних кроків:

1. Конвертація YOLOv8 до TFLite: модель yolov8n.pt була перетворена у .tflite з float32-вагами через бібліотеку Ultralytics[26].

2. TFLiteModelHelper: реалізує передобробку (масштабування, нормалізацію), запуск інференції та зчитування тензорів[27].

3. Non-Maximum Suppression (NMS): фільтрує результати з перекриттям, залишаючи лише найрелевантніші об’єкти (рис. 3.15).

4. Інтеграція в ImageAnalyzer: кадри з камери обробляються у реальному часі. На середньостатистичних пристроях досягається частота до 10 fps без видимих затримок.

Веб-компонент: особливості та можливість масштабування

- Веб-інтерфейс — повністю вбудований у застосунок як статичний ресурс. Може бути розгорнутий окремо на веб-сервері або CDN (наприклад, Firebase Hosting), що дозволяє передавати потік через WebRTC або RTMP.
- Масштабування: для розширення системи можливе винесення обробки в браузер або сервер, інтеграція з хмарними службами, або використання CDN для розподілу навантаження в багатокористувацьких сценаріях.

Формування PDF-звітів. Однією з функцій системи є формування PDF-звітів із виявленими фактами куріння. Процес реалізовано у застосунку засобами Android.

Формування звіту включає:

- автоматичне створення таблиці подій із датою, часом, класом об’єкта, рівнем впевненості;
- вбудовування скріншотів із bounding boxes;
- збереження PDF у локальне сховище.

Звіти можуть бути використані як доказ порушення правил або архів спостережень за певний період. Ця функція важлива для навчальних закладів, офісів, підприємств та охоронних структур (рис. 3.16).

Для публікації в Google Play Store застосунок адаптується до вимог платформи:

- декларуються дозволи на використання камери та збереження файлів;
- додаються політика конфіденційності та повідомлення про обробку персональних даних;
- проходить перевірка на безпеку, сумісність і відповідність Google Guidelines.
- Завдяки автономності, низьким вимогам до ресурсів та простоті розгортання, система підходить для використання:
- у школах та університетах;
- в офісах, лабораторіях, виробничих приміщеннях;
- на приватних об’єктах для контролю порушень.

Результати тренування обраної моделі YOLOv8n демонструють стабільну динаміку зниження функцій втрат як на тренувальних, так і на валідаційних даних[28]. Аналогічна тенденція простежується і на графіках val/*_loss, що свідчить про відсутність перенавчання та хорошу узагальнюючу здатність моделі до нових даних. Метрики якості на останній версії моделі з датасетом в 6100 зображень, становлять: mAP@0.5 = 0.904, mAP@0.5:0.95 = 0.62, Precision = 0.877, Recall = 0.820. Це підтверджує високу точність і повноту виявлення об’єктів.

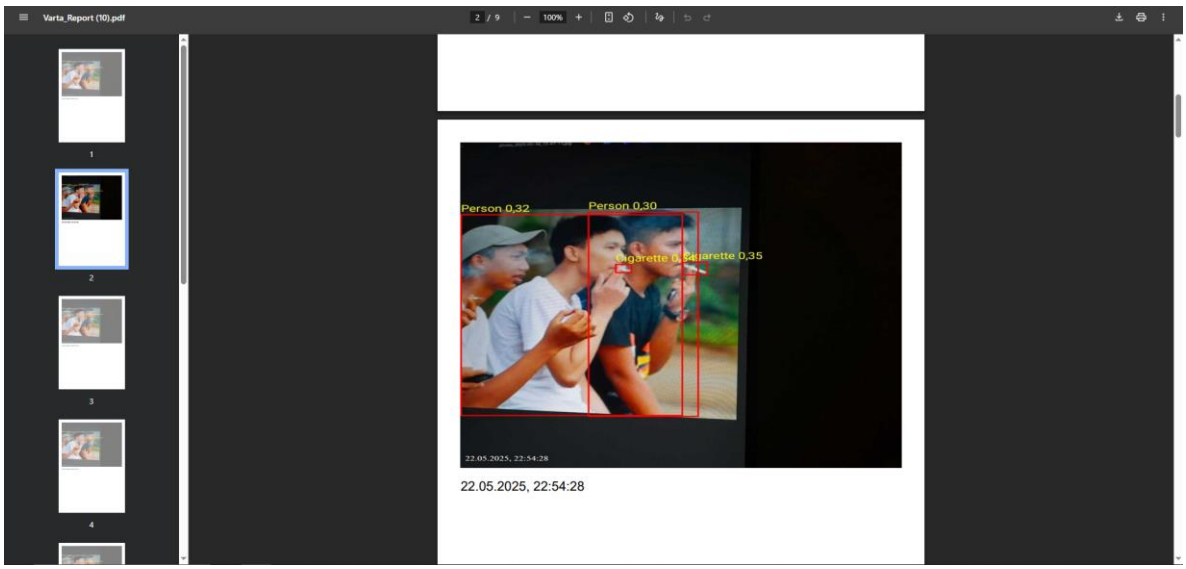


Рисунок 3.16 — Результат експорту PDF-звіту

Модель виявлення об’єктів демонструвала найкращий результат при визначенні об’єктів класу Person, який класифікується стабільно і з високою впевненістю, об’єкти класу Cigarette виявлялись з деякою частиною хибних результатів, а найбільш проблемни є клас vape-pod, враховуючи специфіку даних пристроїв та їх різноманіття, що призводило до хибної класифіковації як background або змішування з класом Cigarette.

Це може бути наслідком:

– візуальної схожості між об'єктами класів vape-pod та Cigarette та їхніми невеликим габаритами

– нестачі репрезентативних прикладів у навчальному наборі для певних ситуацій (частково приховані, незвичні кути огляду тощо).

– Незважаючи на високу загальну точність, аналіз матриць помилок вказує на потенційні напрями подальшого вдосконалення:

– покращення анотацій класу vape-pod,

– підсилення датасету складними прикладами,

– Графіки залежностей Recall, Precision та F1 від порогу впевненості (confidence) дозволили підібрати оптимальне значення порогу для реального використання моделі. Наприклад:

– у задачах, де критично важливо не пропустити випадки куріння, доцільно встановити низький поріг (~0.3–0.4), навіть ціною деякого зростання кількості помилкових спрацювань;

– у випадках, де важливіша максимальна точність, доцільно підняти поріг до 0.8–0.9, щоб уникнути зайвих детекцій;

– для типових сценаріїв (баланс між точністю і повнотою), графік F1 показує, що оптимальний поріг впевненості складає 0.43, при якому досягається максимальне значення F1-метрики (0.83).

Загалом модель YOLOv8n показала високу ефективність при обмеженій кількості параметрів, що дозволяє її інтегрувати в мобільні пристрої без суттєвих втрат у якості. Реалізована система використовує модель у форматі .tflite, оптимізовану для float32, та працює у поєднанні з класом TFLiteModelHelper, який відповідає за масштабування зображень, нормалізацію, запуск інференсу та постобробку результатів через NMS. Інтеграція виконана у модулі ImageAnalyzer, де кадри з камери обробляються в реальному часі.

Таким чином, реалізований мобільний застосунок об'єднує інструменти відеоаналітики, нейромережевого виявлення та веб-взаємодії, не потребуючи підключення до хмарних сервісів. Це робить його придатним для автономного використання у навчальних закладах, медичних установах, підприємствах та інших об'єктах, де необхідний оперативний контроль за дотриманням правил щодо заборони куріння.

## 3.4 Висновки

У цьому розділі був описаний процес розроблення інформаційної системи для виявлення фактів куріння на основі аналізу відео з камери мобільного пристрою. Система реалізована у форматі Android-додатку, що поєднує функції потокового відеоспостереження, виявлення об’єктів за допомогою нейромережі та автоматичної генерації звітів.

Було спроєктовано архітектуру інформаційної системи, створено UML-діаграми основних модулів, розроблено алгоритм роботи мобільного застосунку, а також здійснено його повну програмну реалізацію. У межах застосунку була впроваджена нейромережева модель глибинного навчання у форматі TFLite, що дозволяє здійснювати виявлення об’єктів на пристрої без підключення до мережі Інтернет.

Система включає модуль локального MJPEG-стріму, який транслює відео з камери у браузер через HTTP, модуль обробки кадрів у реальному часі з виявленням сигарет, людей та pod-пристроїв, механізм збереження скріншотів із часовими мітками та формування PDF-звітів про виявлені факти.

Особливу увагу було приділено підготовці та тренуванню нейронної моделі. Початкове опрацювання датасету здійснювалось за допомогою платформи Roboflow. Після тестування інструментом Ultralytics HUB було виявлено певні обмеження — модель експортувалась з підтримкою лише одного класу та надмірною вагою, тому було ухвалено рішення продовжити процес тренування моделі на платформу Kaggle та застосувати модель іншого розміру, а саме YOLO8n, за рахунок чого вдалося досягти оптимальної якості моделі та її подальшого експорту у формат TFLite.

У результаті було створено автономну нейромережеву інформаційну систему, яка поєднує гнучкість мобільного додатку з можливостями глибинного навчання, що дозволяє ефективно використовувати її у реальних умовах моніторингу та фіксації порушень.

# ВИСНОВКИ

У ході виконання бакалаврської кваліфікаційної роботи було здійснено повний цикл розробки нейромережевої інформаційної системи для виявлення фактів куріння на основі відеопотоку з мобільного пристрою. Актуальність теми обґрунтована поширеністю тютюнопаління серед молоді та необхідністю автоматизації контролю за дотриманням правил у громадських місцях. Проаналізовано сучасні підходи до розв'язання подібних задач за допомогою глибинного навчання, зокрема моделей об’єктного виявлення у мобільній відеоаналітиці.

Для реалізації системи було обрано мову програмування Kotlin, середовище Android Studio та платформу TensorFlow Lite, що забезпечило створення ефективного, автономного застосунку без потреби у зовнішніх сервісах. У рамках проєкту розроблено кастомний датасет із понад 6100 зображень, що охоплює класи "людина", "сигарета" та "vape-пристрій". Навчання моделі YOLOv8n здійснено за допомогою інструментів Roboflow, Ultralytics HUB та Kaggle, з подальшою конвертацією у формат .tflite, оптимізований під мобільні пристрої.

Результати тренування свідчать про стабільне зниження функцій втрат та відсутність перенавчання. Досягнуті фінальні метрики моделі становлять: mAP@0.5 = 0.904, mAP@0.5:0.95 = 0.62, Precision = 0.877, Recall = 0.820, що підтверджує високу точність і повноту виявлення. Найкращі результати спостерігаються для класу Person, тоді як для Cigarette іноді фіксуються хибні спрацювання, а найбільші труднощі викликає клас vape-pod, що зумовлено його візуальною схожістю з сигаретою, невеликими розмірами та недостатньою репрезентативністю у навчальному наборі. Водночас аналіз метрик F1, Precision і Recall за різними порогами впевненості дозволив визначити оптимальне значення 0.43, при якому модель досягає F1 = 0.83. Це забезпечує збалансовану роботу між мінімізацією хибних спрацювань і повнотою виявлення.

Інтеграція моделі в мобільний застосунок реалізована через клас TFLiteModelHelper, який відповідає за масштабування та нормалізацію

зображень, запуск інференції та постобробку результатів за допомогою Non-Maximum Suppression. Уся обробка кадрів відбувається в режимі реального часу у компоненті ImageAnalyzer, що дозволяє досягти до 10 кадрів на секунду навіть на пристроях середнього рівня.

Застосунок також включає вбудований MJPEG-сервер, HTML-інтерфейс, логування подій, генерацію PDF-звітів, а також забезпечує повну автономність, працюючи локально без потреби в Інтернеті. Створене рішення демонструє практичну ефективність поєднання мобільних технологій і комп’ютерного зору, а також має потенціал для подальшого розвитку: додавання бази даних подій, розширення класів виявлення, масштабування веб-компоненту через CDN або хмарні сервіси. Отримані результати підтверджують доцільність застосування таких систем у школах, лікарнях, на підприємствах та в інших місцях, де важливо оперативно реагувати на порушення заборони куріння.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ігор Кузін: медичні попередження про шкоду куріння займатимуть 65% площі пачки — це посилить ефект антитютюнових обмежень. URL: https://moz.gov.ua/uk/igor-kuzin-medichni-poperedzhennja-pro-shkodu-kurinnja-zajmatimut-65-ploschi-pachki---ce-posilit-efekt-antitjutjunovih-obmezhen (дата звернення: 28.05.2025).

2. Як змінювалася кількість курців в Україні та світі. URL: https://life.pravda.com.ua/health/2023/05/31/254563 (дата звернення: 28.05.2025).

3. Результати опитування: як в Україні змінюється споживання тютюнових та нікотинових виробів. URL: https://center-life.org/opituvannja-tyutyunovi-virobi (дата звернення: 28.05.2025).

4. Кожен п’ятий підліток в Україні курить електронні сигарети. URL: https://moz.gov.ua/article/news/kozhen-p-jatij-pidlitok-v-ukraini-kurit-elektronni-sigareti (дата звернення: 28.05.2025).

5. Ultralytics. YOLOv8 Documentation. URL: https://docs.ultralytics.com (дата звернення: 28.05.2025).

6. Wang Z., Lei L., Shi P. Smoking behavior detection algorithm based on YOLOv8-MNC Frontiers in Computational Neuroscience. 2023. Vol. 17. DOI: https://doi.org/10.35377/saucis...1461268

7. Kaggle. Detect People Smoking. URL: https://www.kaggle.com/code/ruisantos/detect-people-smoking (дата звернення: 28.05.2025).

8. Kaggle. Smoking Detection | Acc: 89.6%. URL: https://www.kaggle.com/code/abdallahwagih/smoking-detection-acc-896 (дата звернення: 28.05.2025).

9. Android Developers. Kotlin Language Guide. URL: https://developer.android.com/kotlin (дата звернення: 28.05.2025).

10. Android Developers. CameraX Documentation. URL: https://developer.android.com/training/camerax (дата звернення: 28.05.2025).

11. Gradle User Guide. URL: https://docs.gradle.org/current/userguide/userguide.html (дата звернення: 28.05.2025).

12. Android Debug Bridge (ADB). URL: https://developer.android.com/tools/adb (дата звернення: 28.05.2025).

13. NanoHTTPD – a Tiny Web Server in Java. URL: https://github.com/NanoHttpd/nanohttpd (дата звернення: 28.05.2025).

14. Android Developers Hub. URL: https://developer.android.com/develop (дата звернення: 28.05.2025).

15. TensorFlow Lite Guide. URL: https://www.tensorflow.org/lite (дата звернення: 28.05.2025).

16. YOLOv8: Comprehensive Guide to State of the Art Object Detection. URL: https://blog.roboflow.com/yolov8-guide (дата звернення: 28.05.2025).

17. Ultralytics. YOLOv5 usage examples. URL: https://docs.ultralytics.com/models/yolov5/#usage-examples (дата звернення: 28.05.2025).

18. Bakhytov Y., Öz C. Cigarette Detection in Images Based on YOLOv8 Sakarya University Journal of Computer and Information Sciences. 2024. Vol. 7. DOI: https://doi.org/10.35377/saucis.2024.7.1.1461268

19. Roboflow Platform. URL: https://roboflow.com/ (дата звернення: 28.05.2025).

20. Ultralytics Hub. URL: https://hub.ultralytics.com/ (дата звернення: 28.05.2025).

21. Kaggle. URL: https://www.kaggle.com/ (дата звернення: 28.05.2025).

22. Мокін В. Б., Дратований М. В. Інтелектуальний метод з підкріпленням синтезу оптимального конвеєру операцій попереднього оброблення даних у задачах машинного навчання. Наукові праці ВНТУ. 2022. № 4. С. 15–25. DOI: https://doi.org/10.31649/2307-5376-2022-4-15-25

23. Знаміровський А. О., Лосенко А. В. Інформаційна технологія виявлення курців у відеопотоці з використанням нейромережевих моделей // Міжнародна науково-практична інтернет-конференція студентів, аспірантів та

молодих науковців «Молодь в науці: дослідження, проблеми, перспективи» (2025). Вінниця : ВНТУ, 2025. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25205 (дата звернення: 10.06.2025).

24. YOLO: Algorithm for Object Detection Explained. URL: https://learnopencv.com/yolo-object-detection (дата звернення: 28.05.2025).

25. Мокін В. Б., Дратований М. В. Наука про дані: машинне навчання та інтелектуальний аналіз даних : електронний навчальний посібник комбінованого (локального та мережевого) використання. – Вінниця : ВНТУ, 2024. 263 с.

26. Kaggle. Varta Object Detection. URL: https://www.kaggle.com/code/deltachhh/varta-object-detection-diploma (дата звернення: 28.05.2025).

27. Ultralytics Python Package. URL: https://pypi.org/project/ultralytics (дата звернення: 28.05.2025).

28. Штовба С. Д., Козачко О. М. Machine learning: стартовий курс : електронний навчальний посібник. Вінниця : ВНТУ, 2020. 81 с.

Додаток А

(обов’язковий)

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації

ЗАТВЕРДЖУЮ

Завідувач кафедри САІТ

_____ д.т.н., проф. Віталій МОКІН

“____”_____ 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на бакалаврську кваліфікаційну роботу

НЕЙРОМЕРЕЖЕВА ІНФОРМАЦІЙНА СИСТЕМА ВИЯВЛЕННЯ КУРЦІВ НА ОСНОВІ ПОТОКОВОГО ВІДЕО

08-53.БКР.007.00.000 ТЗ

Керівник: PhD., асистент. каф. САІТ

_____ Арсен ЛОСЕНКО

«__» _____ 2025 р.

Розробив студент гр. 2ІСТ-21б

_____Артем ЗНАМІРОВСЬКИЙ

«__» _____ 2025 р.

Вінниця 2025

1. Підстава для проведення робіт.

Підставою для виконання роботи є наказ №__ по ВНТУ від «__» _____2025р., та індивідуальне завдання на БКР, затверджене протоколом №__ засідання кафедри САІТ від «__» _____ 2025р.

2. Джерела розробки:

1) TensorFlow Lite Guide. URL: https://www.tensorflow.org/lite (дата звернення: 28.05.2025).

2) Ultralytics Hub. URL: https://hub.ultralytics.com/ (дата звернення: 28.05.2025).

3. Мета і призначення роботи.

Метою дослідження є розроблення інформаційної системи виявлення курців у потоковому відео з використанням нейромережевої моделі виявлення об’єктів.

4. Вихідні дані для проведення робіт:

Датасет анотованих зображень курців для тренування нейромережові моделі, датасет показників ефективності моделей машинного навчання

5. Методи дослідження:

Дослідження існуючих інформаційних систем, тренування моделі YOLO, розробка UML-діаграм, створення макетів мобільного застосунку, реалізація REST API на Kotlin, клієнт-серверна взаємодія через HTTP-запити, тестування застосунку.

6. Етапи роботи і терміни їх виконання:

a) Аналіз предметної області _____ – _____
b) Порівняльний аналіз проблематики _____ – _____
c) Вибір оптимальних інформаційних технологій _____ – _____
d) Розроблення інформаційної системи виявлення курців _____ – _____
e) Оформлення матеріалів до захисту БКР _____ – _____

7. Очікувані результати та порядок реалізації

Отримання інформаційної системи виявлення курців на основі потокового відео за допомогою нейронних мереж.

8. Вимоги до розробленої документації

Текстова та ілюстративна частини роботи оформлені у відповідності до вимог «Методичних вказівок до виконання бакалаврських кваліфікаційних робіт для студентів спеціальностей: 124 «Системний аналіз», 126 «Інформаційні системи та технології» (освітня програма «Прикладні інформаційні технології»)».

9. Порядок приймання роботи

Публічний захист «___» _____ 2025 р.
Початок розробки «___» _____ 2025 р.
Граничні терміни виконання БКР «___» _____ 2025 р.

Розробив студент групи 2ІСТ-21б _____Артем ЗНАМІРОВСЬКИЙ

Додаток Б

(обов’язковий)

**ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ**

Назва роботи: «Нейромережева інформаційна система виявлення курців на основі потокового відео»

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ: кафедра САІТ, ФІІТА, гр. 2ІСТ-21б

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,99 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне):

- ■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- □ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- □ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Віталій МОКІН, зав. каф. САІТ _____
(підпис)

Євгеній КРИЖАНОВСЬКИЙ, доц. каф. САІТ _____
(підпис)

Особа, відповідальна за перевірку _____ Сергій ЖУКОВ
(підпис)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Арсен ЛОСЕНКО, PhD, ас. каф. САІТ
(підпис)

Здобувач _____ Артем ЗНАМІРОВСЬКИЙ

# Додаток В

(довідниковий)

## Фрагмент лістингу програми

Код файлу MainActivity.kt

```
package com.example.mjpegstreamer

import android.Manifest
import android.content.pm.PackageManager
import android.graphics.*
import android.os.Bundle
import android.os.Handler
import android.util.Log
import android.widget.Button
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import androidx.camera.core.*
import androidx.camera.lifecycle.ProcessCameraProvider
import androidx.camera.view.PreviewView
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import java.io.ByteArrayOutputStream
import java.net.Inet4Address
import java.net.NetworkInterface
import java.util.concurrent.Executors
import kotlin.math.abs

class MainActivity : AppCompatActivity() {

    private lateinit var previewView: PreviewView
    private lateinit var startBtn: Button
    private lateinit var ipText: TextView
    private lateinit var statusText: TextView

    private lateinit var detectionOverlay: DetectionOverlay
    private lateinit var modelHelper: TFLiteModelHelper
    private lateinit var frameAnalyzer: FrameAnalyzer

    private var server: MJPEGServer? = null
    private var isStreaming = false
    private val cameraExecutor = Executors.newSingleThreadExecutor()
    private var lastFrame: ByteArray? = null

    companion object {
        @Volatile var useFrontCamera = false
        @Volatile var switchRequested = false
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
        setContentView(R.layout.activity_main)

        previewView = findViewById(R.id.previewView)
        startBtn = findViewById(R.id.startButton)
        ipText = findViewById(R.id.ipTextView)
        statusText = findViewById(R.id.statusTextView)

        ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.CAMERA,
Manifest.permission.INTERNET), 1)

        modelHelper = TFLiteModelHelper(this)
        frameAnalyzer = FrameAnalyzer(this, modelHelper)
        detectionOverlay = DetectionOverlay(this)
        addContentView(detectionOverlay, previewView.layoutParams)

        startBtn.setOnClickListener {
            toggleStreaming()
        }

        startSwitchWatcher()
    }

    private fun toggleStreaming() {
        isStreaming = !isStreaming
        if (isStreaming) {
            startCamera()
            server = MJPEGServer().apply { startServer() }
            ipText.text = "http://${getLocalIpAddress()}:8080"
            statusText.text = "\uD83D\uDCF1 Streaming..."
            startBtn.text = "Stop Streaming"
        } else {
            server?.stop()
            server = null
            statusText.text = "\uD83D\uDD34 Not streaming"
            startBtn.text = "Start Streaming"
            ipText.text = "IP will appear here"
        }
    }

    private fun getLocalIpAddress(): String {
        return try {
            NetworkInterface.getNetworkInterfaces().asSequence().flatMap {
it.inetAddresses.asSequence() }
                .firstOrNull { !it.isLoopbackAddress && it is Inet4Address }?.hostAddress ?: "127.0.0.1"
        } catch (ex: Exception) {
            ex.printStackTrace()
            "127.0.0.1"
        }
    }

    private fun startCamera() {
        val cameraProviderFuture = ProcessCameraProvider.getInstance(this)
        cameraProviderFuture.addListener({
            val cameraProvider = cameraProviderFuture.get()
```

```
        if (switchRequested) {
            useFrontCamera = !useFrontCamera
            switchRequested = false
        }

        val preview = Preview.Builder().build().also {
            it.setSurfaceProvider(previewView.surfaceProvider)
        }

        val imageAnalyzer = ImageAnalysis.Builder()
            .setTargetResolution(android.util.Size(640, 480))
            .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
            .build()

        imageAnalyzer.setAnalyzer(cameraExecutor) { imageProxy ->
            try {
                val yPlane = imageProxy.planes[0].buffer
                val ySize = yPlane.remaining()
                val currentFrame = ByteArray(ySize).apply { yPlane.get(this) }
                val motion = lastFrame?.let { compareFrames(it, currentFrame) } ?: false
                lastFrame = currentFrame
                if (motion) MJPEGServer.motionDetected = true

                val nv21 = convertToNV21(imageProxy, currentFrame)
                val yuvImage = YuvImage(nv21, ImageFormat.NV21, imageProxy.width,
imageProxy.height, null)
                val out = ByteArrayOutputStream().apply {
                    yuvImage.compressToJpeg(Rect(0, 0, imageProxy.width, imageProxy.height), 50,
this)
                }
                val jpegData = out.toByteArray()
                val bitmap = BitmapFactory.decodeByteArray(jpegData, 0,
jpegData.size).copy(Bitmap.Config.ARGB_8888, true)

                val results = frameAnalyzer.analyzeJpegFrame(jpegData)
                val canvas = Canvas(bitmap)
                val paint = Paint().apply {
                    color = Color.RED
                    style = Paint.Style.STROKE
                    strokeWidth = 4f
                }
                val textPaint = Paint().apply {
                    color = Color.YELLOW
                    textSize = 32f
                    style = Paint.Style.FILL
                }
                results.forEach {
                    canvas.drawRect(it.rect, paint)
                    canvas.drawText("${it.label} %.2f".format(it.confidence), it.rect.left, it.rect.top - 10,
textPaint)
                }

                ByteArrayOutputStream().apply {
                    bitmap.compress(Bitmap.CompressFormat.JPEG, 50, this)
```

```
                MJPEGStream.sendJpegFrame(toByteArray())
            }
        } catch (e: Exception) {
            Log.e("CameraAnalyzer", "Error: ${e.message}")
        } finally {
            imageProxy.close()
        }
    }

    try {
        cameraProvider.unbindAll()
        val cameraSelector = if (useFrontCamera)
CameraSelector.DEFAULT_FRONT_CAMERA else
CameraSelector.DEFAULT_BACK_CAMERA
        cameraProvider.bindToLifecycle(this, cameraSelector, preview, imageAnalyzer)
        Log.d("CameraX", "✅ Camera bound")
    } catch (e: Exception) {
        Log.e("CameraX", "🚨 Camera bind failed", e)
    }
}, ContextCompat.getMainExecutor(this))
}

private fun convertToNV21(imageProxy: ImageProxy, currentFrame: ByteArray): ByteArray {
    val uPlane = imageProxy.planes[1].buffer
    val vPlane = imageProxy.planes[2].buffer
    val ySize = currentFrame.size
    val nv21 = ByteArray(ySize + uPlane.remaining() + vPlane.remaining()).apply {
        System.arraycopy(currentFrame, 0, this, 0, ySize)
    }
    var offset = ySize
    val rowStride = imageProxy.planes[1].rowStride
    val pixelStride = imageProxy.planes[1].pixelStride
    for (row in 0 until imageProxy.height / 2) {
        for (col in 0 until imageProxy.width / 2) {
            val vuIndex = row * rowStride + col * pixelStride
            nv21[offset++] = vPlane.get(vuIndex)
            nv21[offset++] = uPlane.get(vuIndex)
        }
    }
    return nv21
}

private fun compareFrames(frame1: ByteArray, frame2: ByteArray, threshold: Int = 20): Boolean
{
    if (frame1.size != frame2.size) return false
    val diff = (frame1.indices step 10).sumOf { abs(frame1[it].toInt() - frame2[it].toInt()) }
    return diff / (frame1.size / 10) > threshold
}

private fun startSwitchWatcher() {
    Handler(mainLooper).post(object : Runnable {
        override fun run() {
            if (switchRequested && isStreaming) startCamera()
```

```
            Handler(mainLooper).postDelayed(this, 1000)
        }
    })
  }
}
```

Код файлу TFLiteModelHelper.kt

```
package com.example.mjpegstreamer

import android.content.Context
import android.graphics.Bitmap
import android.graphics.RectF
import android.util.Log
import org.tensorflow.lite.Interpreter
import java.io.File
import java.nio.ByteBuffer
import java.nio.ByteOrder
import kotlin.math.exp
import kotlin.math.max
import kotlin.math.min

class TFLiteModelHelper(context: Context) {

    private val interpreter: Interpreter
    private val labels = listOf("Person", "Cigarette", "vape-pod")
    private val inputSize = 640
    private val confThreshold = 0.30f
    private val iouThreshold = 0.5f

    init {
        val modelFile = File(context.filesDir, "diplom.tflite")
        if (!modelFile.exists()) {
            context.assets.open("diplom.tflite").use { input ->
                modelFile.outputStream().use { output -> input.copyTo(output) }
            }
        }
        interpreter = Interpreter(modelFile)
    }

    fun runInference(bitmap: Bitmap): List<DetectionResult> {
        val resized = Bitmap.createScaledBitmap(bitmap, inputSize, inputSize, true)
        val inputBuffer = ByteBuffer.allocateDirect(1 * inputSize * inputSize * 3 * 4)
        inputBuffer.order(ByteOrder.nativeOrder())
        inputBuffer.rewind()

        for (y in 0 until inputSize) {
            for (x in 0 until inputSize) {
                val px = resized.getPixel(x, y)
                inputBuffer.putFloat(((px shr 16) and 0xFF) / 255.0f)
                inputBuffer.putFloat(((px shr 8) and 0xFF) / 255.0f)
                inputBuffer.putFloat((px and 0xFF) / 255.0f)
            }
        }
```

```
        val rawOutput = Array(1) { Array(7) { FloatArray(8400) } }
        interpreter.run(inputBuffer, rawOutput)

        val transposed = Array(8400) { FloatArray(7) }
        for (i in 0 until 7) {
            for (j in 0 until 8400) {
                transposed[j][i] = rawOutput[0][i][j]
            }
        }

        val detections = mutableListOf<DetectionResult>()

        for (row in transposed) {
            val cx = row[0]
            val cy = row[1]
            val w = row[2]
            val h = row[3]
            val objectness = sigmoid(row[4])
            val classScores = row.sliceArray(5..6).map { sigmoid(it) }
            val classId = classScores.indices.maxByOrNull { classScores[it] } ?: continue
            val classScore = classScores[classId]
            val finalScore = objectness * classScore

            if (finalScore > confThreshold && classId < labels.size) {
                val rect = RectF(
                    (cx - w / 2) * bitmap.width,
                    (cy - h / 2) * bitmap.height,
                    (cx + w / 2) * bitmap.width,
                    (cy + h / 2) * bitmap.height
                )

                detections.add(DetectionResult(labels[classId], finalScore, rect))
                Log.d("TFLite", "✅ Detected: ${labels[classId]} ${"%.2f".format(finalScore)} at
$rect")
            }
        }

        Log.d("TFLite", "🧱 Total raw detections: ${detections.size}")
        val finalResults = applyNMS(detections)
        Log.d("TFLite", "🧹 After NMS: ${finalResults.size}")
        return finalResults
    }

    private fun sigmoid(x: Float): Float {
        return (1.0 / (1.0 + exp(-x))).toFloat()
    }

    private fun applyNMS(detections: List<DetectionResult>): List<DetectionResult> {
        val sorted = detections.sortedByDescending { it.confidence }.toMutableList()
        val results = mutableListOf<DetectionResult>()

        while (sorted.isNotEmpty()) {
            val best = sorted.removeAt(0)
```

```
            results.add(best)
            sorted.removeAll { other ->
                iou(best.rect, other.rect) > iouThreshold
            }
        }
        return results
    }

    private fun iou(a: RectF, b: RectF): Float {
        val ax1 = a.centerX() - a.width() / 2
        val ay1 = a.centerY() - a.height() / 2
        val ax2 = a.centerX() + a.width() / 2
        val ay2 = a.centerY() + a.height() / 2

        val bx1 = b.centerX() - b.width() / 2
        val by1 = b.centerY() - b.height() / 2
        val bx2 = b.centerX() + b.width() / 2
        val by2 = b.centerY() + b.height() / 2

        val interX1 = max(ax1, bx1)
        val interY1 = max(ay1, by1)
        val interX2 = min(ax2, bx2)
        val interY2 = min(ay2, by2)

        val interArea = max(0f, interX2 - interX1) * max(0f, interY2 - interY1)
        val areaA = (ax2 - ax1) * (ay2 - ay1)
        val areaB = (bx2 - bx1) * (by2 - by1)
        val union = areaA + areaB - interArea

        return if (union > 0f) interArea / union else 0f
    }
}
```

Додаток Г

(обов’язковий)

**ІЛЮСТРАТИВНА ЧАСТИНА**

НЕЙРОМЕРЕЖЕВА ІНФОРМАЦІЙНА СИСТЕМА ВИЯВЛЕННЯ КУРЦІВ НА ОСНОВІ ПОТОКОВОГО ВІДЕО

Нормоконтроль: к.т.н., доцент

_____ Сергій ЖУКОВ

«___» _____ 2025 р.

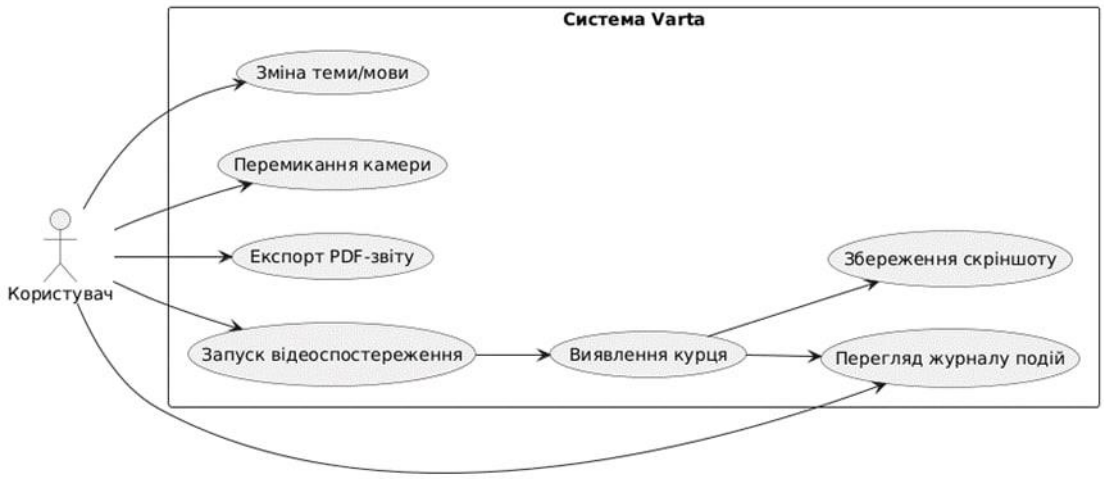


Рисунок Г.1 – Діаграма використання мобільного застосунку

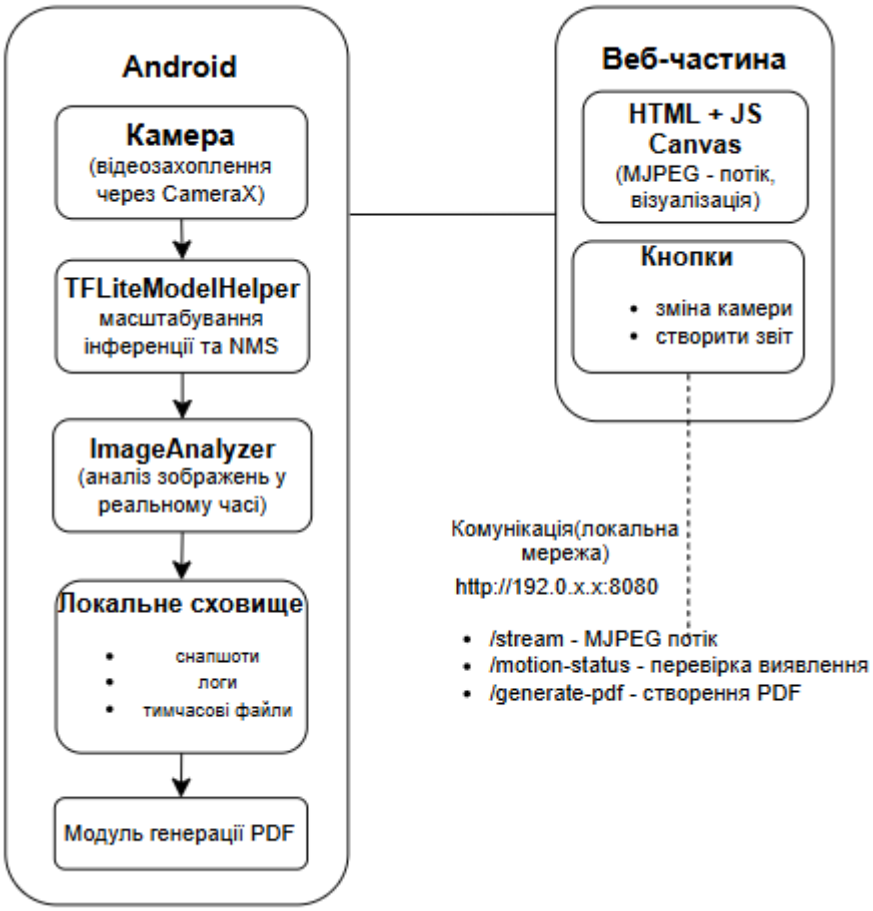


Рисунок Г.2 – Архітектура інформаційної системи

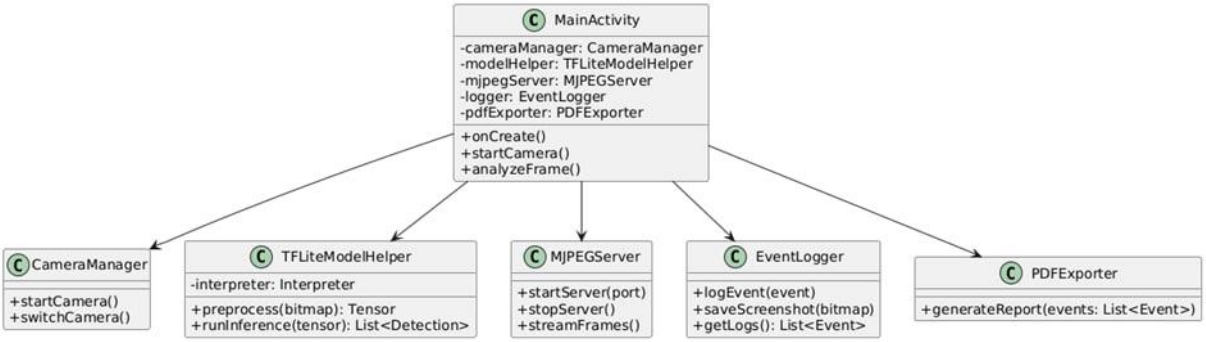


Рисунок Г.3 – Діаграма класів інформаційної системи

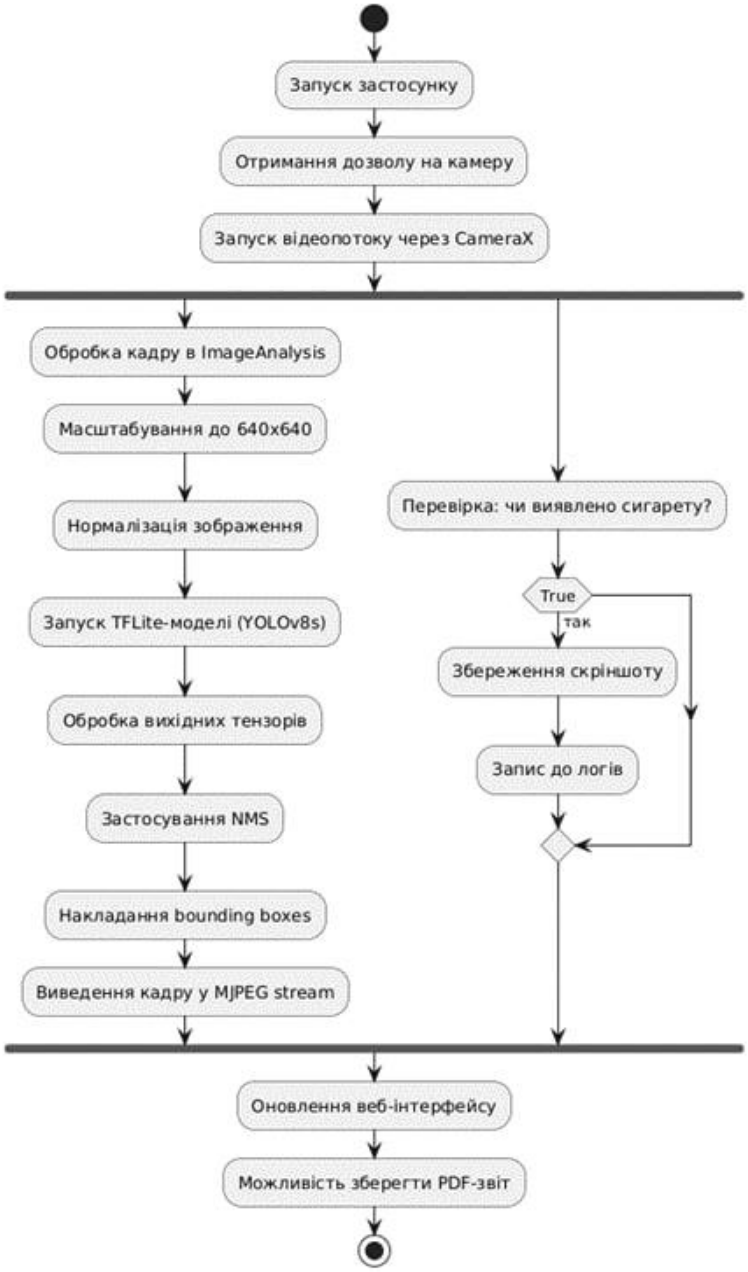


Рисунок Г.4 – UML-діаграма діяльності мобільного застосунку

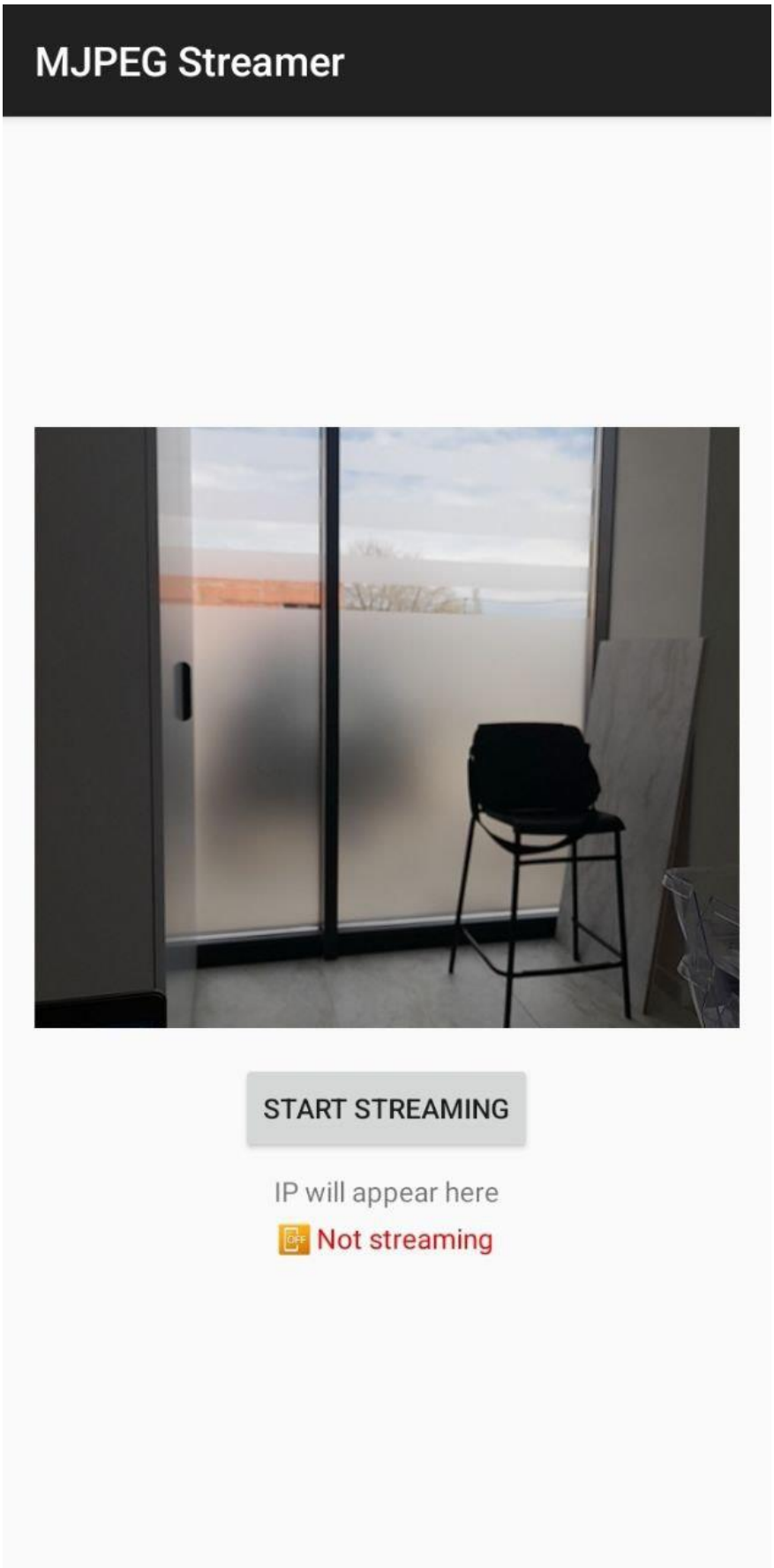


Рисунок Г.5 – Інтерфейс мобільного застосунку

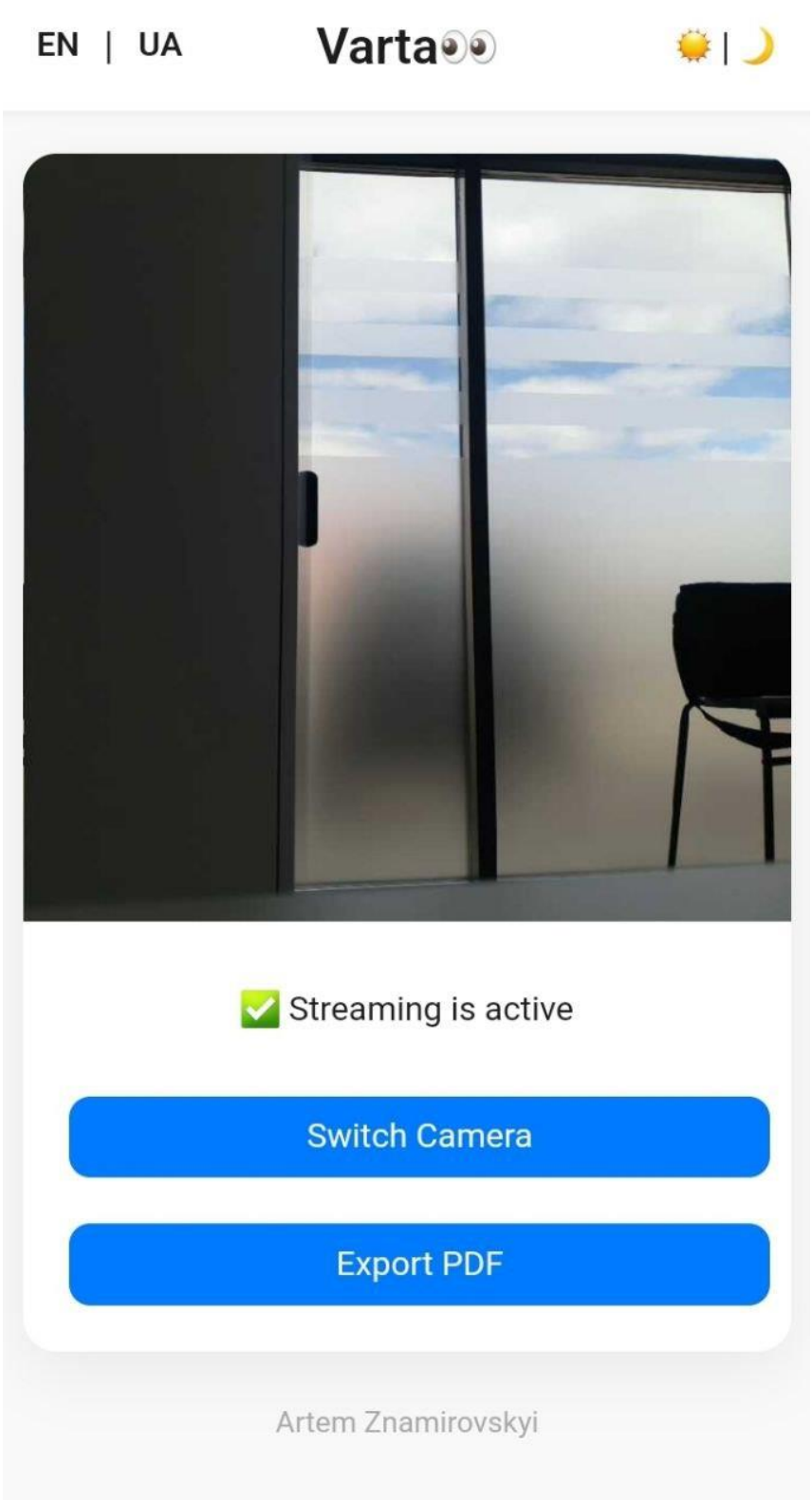


Рисунок Г.6 – Інтерфейс вебклієнту

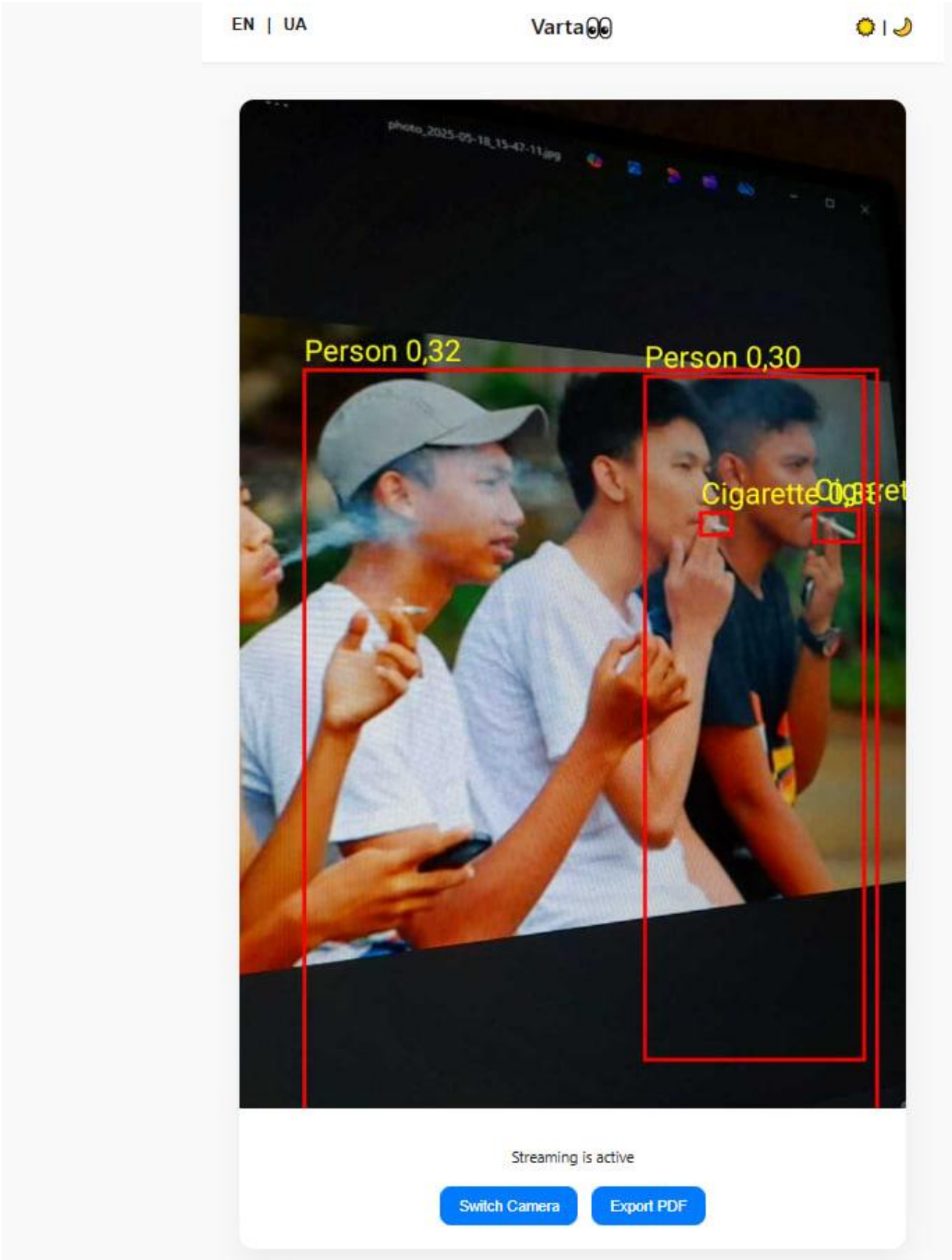


Рисунок Г.7 – Робота нейромережі у вебінтерфейсі