МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

_____

(підпис)

18 грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Інформаційна технологія розпізнавання об'єктів на зображенні в умовах ресурсних обмежень»

здобувача групи ІН.м - 22 Колісник Олексій Миколайович

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

_____ Олексій КОЛІСНИК

(підпис)

Керівник,
старша викладачка кафедри
комп'ютерних наук, кандидат
технічних наук _____ Альона МОСКАЛЕНКО

(підпис)

Суми – 2023

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

_____

(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»

здобувача групи ІН.м-22 Колісник Олексій Миколайович

1. Тема роботи: «Інформаційна технологія розпізнавання об'єктів на зображенні в умовах ресурсних обмежень»
затверджую наказом по СумДУ від *«06» грудня 2023 р. № 1412-VI*

2. Термін здачі здобувачем кваліфікаційної роботи *до 18 грудня 2023 року*

_____

3. Вхідні дані до кваліфікаційної роботи _

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми та постановка задачі. 2) Огляд методів та моделей для побудови інформаційної технології розпізнавання об'єктів на зображенні в умовах ресурсних обмежень. 3) Розробка інформаційної системи розпізнавання об'єктів на зображенні в умовах ресурсних обмежень. 4) Аналіз результатів.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

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

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

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми*		
2	*Формування і постановка задачі*		
3	*Підбір моделей та методів для побудови інформаційної технології розпізнавання об'єктів на зображенні в умовах ресурсних обмежень*		
4	*Реалізація інформаційної системи розпізнавання об'єктів на зображенні в умовах ресурсних обмежень*		
5	*Аналіз отриманих результатів*		
6	*Оформлення пояснювальної записки*		

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

АНОТАЦІЯ

Записка: 54 стор., 43 рис., 1 додаток, 20 джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною, оскільки присвячена вирішенню актуальної проблеми нашого часу по розпізнаванню об’єктів на зображеннях при наявності ресурсних обмежень шляхом розробки моделей по виявленню та класифікації предметів з використанням відповідних методів та їх подальшою оптимізацією.

Об’єкт дослідження — процес розпізнавання об’єктів на зображенні при обмежених ресурсах.

Мета роботи — розробка інформаційної технології розпізнавання об'єктів на зображенні в умовах ресурсних обмежень з використанням легких моделей для виявлення та розпізнавання предметів.

Методи дослідження — методи побудови моделей для виявлення та класифікації об’єктів з використанням методів підвищення швидкодії.

Результати — розроблено інформаційну технологію, яка зчитує набір зображень з певними об’єктами на них, проводить виявлення об’єктів, вирізає знайдені об’єкти та передає їх далі на класифікацію. Після проходження класифікації, на кожному початковому зображенні будується рамка та надпис з класом для кожного знайденого та класифікованого об’єкту, після чого всі зображення зберігаються у папці з результатами роботи. Проведено тестування на реалістичних наборах тестових даних.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, РЕСУРСНІ ОБМЕЖЕННЯ,
РОЗПІЗНАВАННЯ ОБ’ЄКТІВ, PYTHON, TENSORFLOW

ЗМІСТ

ВСТУП

Інформаційні технології розпізнавання об'єктів на зображеннях в умовах обмежених ресурсів стають ключовим елементом сучасного цифрового світу. У світі, де обробка великих обсягів даних та високопродуктивне розпізнавання об'єктів відіграють вирішальну роль у багатьох сферах життя, необхідність працездатних та ефективних технологій у ситуаціях обмежених обчислювальних ресурсів стає доволі актуальною.

Оптимізація моделей для розпізнавання об'єктів на зображеннях у контексті ресурсних обмежень відкриває нові горизонти в області штучного інтелекту. Вирішуючи завдання точності та швидкодії в умовах, де обчислювальні можливості та обсяг доступної пам'яті обмежені, інформаційні технології для розпізнавання об'єктів становляться основою для впровадження штучного інтелекту у мобільні пристрої, вбудовані системи та інші області, де ефективність та точність відіграють таку ж важливу роль як і об’єм необхідних ресурсних потужностей.

У цьому контексті розвиток легких моделей, методів оптимізації та гібридних підходів до розпізнавання об'єктів відіграє важливу роль у створенні ефективних рішень, які забезпечують не лише високу точність, але й швидке виконання завдань на пристроях з обмеженими ресурсами. Використання таких технологій відкриють нові можливості для розвитку систем спостереження, систем виявлення, систем безпеки та багатьох інших сфер, де ресурси можуть бути обмеженими, а вимоги до точності та швидкості невисокими.

Метою роботи є створення інформаційної технології для розпізнавання об’єктів на зображеннях в умовах ресурсних обмежень. Для цього необхідно розібратися в основах роботи з розпізнаванням предметів на зображеннях та з методами та практиками, які можна впровадити для зменшення ресурсних затрат при незначному зменшенні точності розпізнавання.

Завданням даної роботи є розробка інформаційної технології розпізнавання об'єктів на зображенні в умовах ресурсних обмежень, для можливості ефективного розпізнавання предметів на пристроях з ресурсними обмеженнями чи для економії ресурсів при обробці великих об’ємів даних.

1. АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

Розпізнавання об'єктів на зображеннях у наш час стає все більш актуальною задачею, особливо з урахуванням значного росту доступності та кількості даних і розвитку штучного інтелекту. Однак, цей процес може вимагати вкрай великих обсягів ресурсів, це виникає через необхідність обробити великий обсяг даних та обчислювальні вимоги алгоритмів, що використовуються для розпізнавання. Через розвиток сучасних технологій підвищення якості зображення можуть бути доволі великими за обсягом і через це потребуватимуть значних ресурсів, як для їх зберігання, так і подальшої обробки. В той же час для розпізнавання об'єктів потрібні якісні дані для тренування моделей.

На даний момент існує велика кількість методів розпізнавання об'єктів, кожен з яких має свої переваги та недоліки. У цьому випадку вибір оптимального алгоритму для умов наявного обмеження ресурсів є великим викликом. Умови обмеження ресурсів по обчислювальним можливостям чи за обсягом виділеної пам'яті потребують розробки ефективних та дієвих алгоритмів, що працюватимуть оптимально навіть при наявності умов обмежених ресурсів.

1.1 Сучасний стан та тенденція розвитку моделей і методів детектування об'єктів на зображеннях

Теперішній стан у сфері детектування об'єктів на зображеннях характеризується значними досягненнями в процесі глибинного навчання та штучному інтелекті [1], що привели до розвитку потужних моделей та ефективних методів детектування об'єктів. Тож перейдемо до розгляду ключових тенденцій.

Для початку розглянемо **Convolutional neural network (CNN).** Вони виявилися доволі ефективними у завданнях детектування об'єктів на зображеннях. Отже варто перейти до розгляду їх основних представників.

Першим розглянемо представника **YOLO (You Only Look Once)**, Вона представляє собою модель, яка відома своєю швидкістю та точністю. На відміну від значної кількості інших методів виявлення об’єктів на зображеннях, які виконують операцію пошуку та класифікації окремо один від одного, YOLO призначений для виконання обох завдань одночасно. Тобто, вона виконує один прохід по наданому зображенню для виявлення на ньому об'єктів [2], при цьому розділяючи зображення на сітку та передбачаючи клітинки (bounding boxes) і ймовірності присутності представників класів об'єктів у кожній клітинці сітки, побудованої на зображенні.

Далі перейдемо до розгляду **Faster R-CNN**, що представляє собою метод, котрий використовує пропозиції областей (region proposals), це використовується для того, щоб ефективніше визначати регіони зображення, де можуть знаходитися об'єкти класів, а потім проводить їх класифікацію та точно розміщує границі всіх об’єктів (bounding boxes) на цих пропозиціях [3]. Даний метод став еталоном у виявленні об’єктів на зображенні, завдяки його чудовому балансу між точністю та швидкістю, поєднуючи створення пропозицій регіону та класифікацію об’єктів в уніфікованій структурі.

І нарешті перейдемо до розгляду **SSD (Single Shot Multibox Detector)**, що є моделлю, котра генерує прогнози про клас та положення об'єктів в реальному часі, це відбувається шляхом використання певного набору фіксованих розмірів (bounding boxes) на різних рівнях ознак [4]. Тобто дана модель генерує набір обмежувальних рамок, котрі за замовчуванням мають різні пропорції та масштаби на кількох різних шарах. SSD використовує спеціальні карти функцій для виявлення об’єктів, які мають різний розмір та співвідношення сторін. Приклад однієї з архітектур Single Shot Multibox Detector наведений на рис. 1.1.

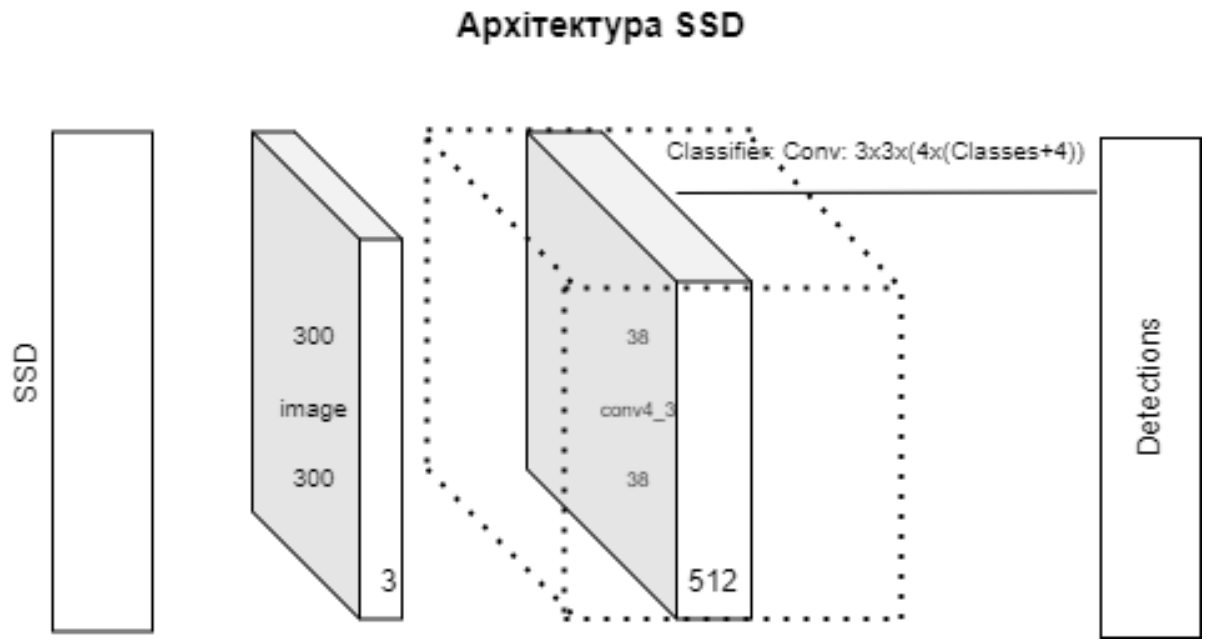


Рисунок 1.1 – Приклад архітектури SSD (Single Shot Multibox Detector)

Далі варто розглянути техніки передбачення об'єктів на різних масштабах. Це є доволі корисним для вирішення нашого завдання по розпізнаванню об’єктів, адже масштаби представлення об’єктів на наших зображеннях очевидно будуть різні. Для нас буде корисним напрямок розробки методів, які здатні передбачати об'єкти на різних масштабах, при чому без значної втрати точності [5]. Це важливо для виявлення об'єктів різних розмірів у складних умовах.

Представником даних технік є пірамідальні архітектури, вони представляють собою використання архітектур, котрі проводять аналіз зображення на різних масштабах одночасно, для того, щоб можна було знаходити об'єкти різних розмірів. Ще одним представником є механізми масштабування функцій активації, фактично це механізми, котрі надають можливість алгоритмам ефективно виявляти об'єкти на різних рівнях деталізації зображення.

Також варто розглянути використання мультимодальних даних, цей процес представляє собою інтеграцію інших джерел даних, наприклад теплову інфрачервону зйомку, та формує нову тенденцію [5]. Це дозволяє системам детектувати об'єкти в умовах, коли звичайне зображення може бути значно

обмежене або взагалі неінформативне. Для побудови нашої системи, застосувати такий підхід не вийде, але це доволі цікава ідея, яка безперечно буде тільки розвиватися.

Тепер перейдемо до розгляду використання архітектур з атенцією, вони представлені такими архітектурами, як наприклад Transformer. В наш час вони починають часто застосовуватися до вирішення завдань детектування об'єктів. Вони дозволяють звертати увагу на важливій області зображення та надають можливість покращити точність детектування.

На останок варто звернути увагу на автоматизоване навчання та оптимізацію гіперпараметрів, дедалі частіше з'являються платформи, котрі надають можливість автоматизувати процес тренування моделей та оптимізації гіперпараметрів, що спрощує розробку моделей детектування об'єктів.

Тож тенденції розвитку спрямовані на подальше поліпшення точності та ефективності детекторів об'єктів, при зменшенні вимог до ресурсів, роботу з мультимодальними даними та розширення можливостей застосування певних алгоритмів у складних умовах. Розвиток цих напрямків сприятиме створенню більш ефективних та універсальних систем розпізнавання об'єктів на зображеннях в різних умовах.

Ці тенденції в моделях та методах детектування об'єктів на зображеннях спрямовані на такі важливі аспекти, як покращення точності виявлення та класифікації, швидкодії та універсальності систем виявлення об'єктів в різноманітних умовах, починаючи від стандартних фотографій до складних умов освітлення чи специфічних класів об'єктів.

1.2 Аналіз методів підвищення швидкодії моделей штучного інтелекту

Для того, щоб провести детальний аналіз розглянемо деякі популярні методи, котрі використовуються для підвищення швидкодії моделей штучного інтелекту.

Першим розглянемо **квантування (Quantization)**, головною його особливістю є те, що воно зменшує точність чисел, які представлені в нейронних мережах, зазвичай зменшуючи бітову точність (наприклад, з 32-бітних чисел до 8-бітних) [6]. Перевага даного підходу полягає в тому, що зменшення кількості бітів призводить до зменшення вимог до пам'яті та операцій, що в свою чергу призводить до підвищення швидкодії обчислень. Водночас головним недоліком є зменшення точності, що може призвести до втрати якості моделі.

Далі ознайомимося з методом **обрізання (Pruning)**: цей метод включає в себе видалення нейронів чи зв'язків між ними, які мають малий вплив на вихідні результати моделі. Головною перевагою даного методу є зменшення кількості параметрів, що в свою чергу дозволяє зменшити необхідний обсяг пам'яті, а також відповідно призводить до збільшення швидкодії проведення обчислень [7]. Недоліком для цього методу є несправедливе обрізання, котре зазвичай може призвести до втрати точності передбачення моделі. Демонстрація роботи даного методу представлена на рис. 1.2.

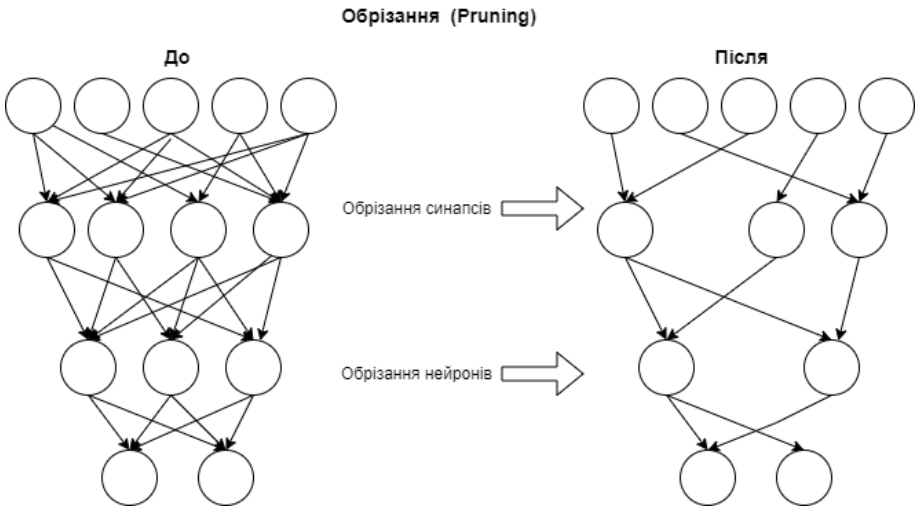


Рисунок 1.2 – Демонстрація роботи Pruning

Цікавим методом є **витягування знань (Knowledge Distillation)**, що в собі передбачає проведення навчання менш складеної моделі на основі знань більш складеної моделі, яку можна назвати вчителем. Головною перевагою даного методу є можливість створення менш складених моделей, які все ще

зберігають більшість знань більш складених моделей. У той же час головним недоліком цього методу є значна залежність від якості наявного вчителя, адже низька якість моделі, яка відіграє роль вчителя буде відображена у створених на його основі моделях, тобто може стати менш ефективним у випадку низької якості вчителя.

Останнім розглянемо **мережі з раннім виходом (Early-Exit Networks),** даний підхід передбачає вихід з моделі при отриманні достатньої інформації, котра необхідна для класифікації. Перевагою цього підходу є те, що він дозволяє економити обчислювальні ресурси, оскільки модель може завершити роботу раніше, якщо вона впевнена у своєму результаті. Недоліки ж у тому, що даний підхід потребує налаштування порогів для прийняття рішення про вихід з моделі; що в свою чергу може впливати на точність отриманого результату.

Отже загалом ці методи призначені для зменшення обчислювальних витрат та обсягів необхідної пам'яті, не дуже суттєво жертвуючи точністю моделей. Вони є ефективними стратегіями для роботи з обмеженими ресурсами, зберігаючи при цьому ефективність та швидкодію моделей штучного інтелекту. Водночас треба обрати саме той метод, що підходить до нашого завдання.

1.3 Формалізована постановка задачі

Постановка задачі для побудови інформаційної технології на зображенні в умовах ресурсних обмежень включає перелік основних завдань, котрі необхідно вирішити:

Для початку необхідно провести зменшення обсягу даних чи приведення даних до оптимального формату, тобто обраний набір даних необхідно по можливості привести до менших розмірів, та залежно від засобів, які будуть використані при навчанні, привести в оптимальний формат. Для цього вони можуть бути стиснуті чи проведені через попередню обробку для скорочення розміру даних.

Для роботи з обмеженими ресурсами мають бути використані легкі моделі, тобто має бути створено чи адаптовано алгоритми розпізнавання, що мають меншу кількість параметрів та вимагають менше обчислювальних ресурсів.

Для покращення роботи моделей мають бути використані техніки оптимізації. Це необхідно тому, що застосування технік оптимізації, таких як квантизація моделей, має зменшити вимоги до пам'яті та обчислювальних можливостей, що має поліпшити можливості роботи наших моделей в умовах ресурсних обмежень.

Також має бути проведена адаптація алгоритмів до специфіки завдання, тобто необхідно розробити чи адаптувати вже існуючі методи, які призначені до конкретного типу об'єктів або умов, щоб підвищити ефективність розпізнавання, для зменшення впливу фактору застосування меншої кількості ресурсів.

Під кінець має бути проведена оцінка метрик продуктивності. Для цього необхідно визначитися з метриками якості виявлення та продуктивності системи для оцінки ефективності в умовах ресурсних обмежень.

Отже загалом для виконання поставленої задачі необхідно підготувати набір даних, який має бути представлений колекцією зображень та міток до них, в котрих відмічено розташування та класифікація об’єктів на зображеннях. На основі цього набору даних необхідно створити модель детектора об’єктів, де всі наявні об'єкти класифікувати одним класом, адже дана модель буде відповідати лише за виявлення об’єктів, а не їх класифікацію.

Після визначення необхідних об’єктів треба провести визначення класу конкретного об'єкта застосувати класифікатор на вирізаних частинах зображеннях на яких знаходяться об'єкти знайдені моделлю детектора, використовуючи більш складну модель з адаптивним екзаменом [8].

Результати роботи системи передбачень будемо зберігати у окрему директорію, для подальшого перегляду та оцінки якості результатів. Окрім цього будемо зберігати оцінки, отримані від метриків.

У результаті виконання розробки маємо отримати модель детектора, яка ефективно розпізнає об'єкти на зображеннях з обмеженими ресурсами, уточнені класи об'єктів після застосування класифікатора з адаптивним екзаменом, ефективне розпізнавання об'єктів на зображеннях з використанням мінімальних обчислювальних ресурсів та збереження адекватної точності розпізнавання класів об'єктів після уточнення за допомогою класифікатора.

2 МОДЕЛІ І МЕТОДИ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ НА ЗОБРАЖЕННІ В УМОВАХ РЕСУРСНИХ ОБМЕЖЕНЬ

Вибір правильних моделей та методів інформаційної технології розпізнавання об'єктів на зображеннях в умовах ресурсних обмежень є ключовими для досягнення ефективності та точності без значного споживання обчислювальних ресурсів.

2.1 Гібридна модель детектора об'єктів з адаптивним режимом класифікації об'єктів

Гібридна модель детектора об'єктів з адаптивним режимом класифікації об'єктів поєднує в собі переваги SSD у вигляді ефективності та точності з методами адаптивної класифікації для ефективного розпізнавання об'єктів умовах обмежених ресурсів.

Детектор об'єктів на базі SSD (Single Shot Multibox Detector) є вельми ефективними у багатьох сценаріях, особливо в умовах обмежених ресурсів [9].

Одна з переваг SSD полягає в тому, що він забезпечує точність виявлення об'єктів, використовуючи при цьому менше обчислювальних ресурсів, що робить його ідеальним вибором для систем з обмеженими ресурсами. Слід зазначити, що модель типу SSD оптимізована та заточена під мінімальне використання ресурсів. Вона працює ефективно на пристроях з обмеженими можливостями, забезпечуючи високу швидкість роботи при низькому споживанні енергії.

Ще однією перевагою є те, що SSD використовує різні шари для виявлення об'єктів різної розмірності на зображеннях, що робить його ефективним у виявленні як невеликих, так і великих об'єктів. Це є корисною особливістю для нашої системи.

Отже, SSD є привабливими через свою ефективність у використанні ресурсів та здатність працювати швидко та ефективно в умовах обмежених ресурсів [10].

Спираючись на усі переваги SSD, описані до цього, для детектора об’єктів має підійти модель типу SSD MobileNet Lite, так як вона поєднує в собі переваги архітектури MobileNet, що відома своєю легкістю та ефективністю, зі здатністю SSD до точного виявлення об'єктів. Це забезпечує оптимальний баланс між швидкістю та продуктивністю. Її ефективна робота в умовах обмежених ресурсів забезпечує точне виявлення об'єктів без великого споживання пам'яті чи обчислювальної потужності [11], що робить SSD MobileNet Lite відмінним вибором для задач на виявлення об'єктів в умовах обмежених ресурсів.

В свою чергу адаптивний режим класифікації об'єктів представляє собою уточнення класу об'єктів з використанням більш складної моделі [4]. Тобто після детектування всіх об'єктів як одного класу, використовується більш складна модель для уточнення класів окремих об'єктів. Для нашого завдання на роль такої моделі має підійти модель типу SSD, через свою ефективність в класифікації. Також до даної моделі для полегшення обчислень варто застосувати квантування.

Як результат, дана модель має забезпечувати гнучкість та ефективність, тобто вона має зберігати SSD ефективність на пристроях з невеликою кількістю пам'яті та обчислювальною потужністю. Ця гібридна модель має можливості використовувати переваги легких моделей для ефективності ресурсів, і водночас використовувати більш складні моделі для уточнення класифікації об'єктів, що призводить до балансу між продуктивністю та точністю.

2.2 Етапи і метод машинного навчання моделі

У даному розділі описано етапи та методи машинного навчання для побудови гібридної моделі детектора об'єктів з адаптивним режимом класифікації:

Першим етапом є збір та підготовка даних - цей етап включає в себе збір даних для навчання моделі. Дані будуть взяті з відкритих джерел або зі спеціалізованих наборів даних. Потім ці дані потрібно підготувати, за необхідності привести їх до одних розмірів та додати чи удосконалити вже існуюче розмічення об’єктів. У нашому випадку дані будуть приведені до розмірності 640x640 пікселів.

Другим етапом роботи з даними є розбиття на тренувальний та тестовий набори. Для оцінки ефективності моделі дані розділяються на тренувальний та тестовий набори, де тренувальний потім теж може бути розбитий на два набори, для покращення ефективності навчання. Тренувальний набір використовується для навчання моделі, тоді як тестовий - для оцінки її точності.

Третім етапом має бути вибір моделі детектора об'єктів. Дуже важливо обрати легку модель, яка може швидко та ефективно виявляти об'єкти на зображеннях. У нашому випадку буде використана SSD MobileNet Lite.

Четвертим етапом є навчання моделі на тренувальних даних. Отже вибрану модель детектора потрібно навчити на тренувальному наборі даних. Під час навчання модель вчиться розпізнавати об'єкти на зображеннях. Варто зазначити, для даної моделі не важливий етап класифікації, тому, щоб навчати дану модель всі тренувальні дані зводяться до одного класу, щоб покращити можливості виявлення об’єктів.

На п’ятому етапу має відбутися створення та навчання моделі для уточнення класів об'єктів, це потрібно для того, щоб після базового детектування об'єктів була створена окрема модель для уточнення класифікації виявлених об'єктів, тобто, для розпізнавання конкретних класів. Для навчання

даної моделі буде використаний той самий навчальний набір даних, що і для попередньої моделі, з тією відмінністю, що у цьому наборі всі дані будуть входити не в один клас, а весь відповідний набір класів, що представлений у цьому наборі даних. Для цієї моделі було вибрано тип моделі SSD.

Шостим етапом має бути оцінка точності та ефективності роботи нашої системи, це включає проведення тестів на тестовому наборі для оцінки точності та ефективності. Для цього етапу будуть використані відповідні метрики, більш детально вони будуть описані в наступному розділі.

Останнім етапом є фінальна оптимізація та налаштування параметрів, тобто після оцінки моделі є можливість вносити корективи та оптимізовані параметри для підвищення ефективності розробленої системи.

Усі ці етапи утворюють послідовний підхід до розробки гібридної моделі для виявлення об'єктів та їх подальшої адаптивної класифікації в умовах обмежених ресурсних можливостей.

2.3 Метрики ефективності детектора об'єктів

Розглянемо детальніше набір метрик та виберемо ефективні для даної системи по виявленню та класифікації об'єктів.

Почнемо розгляд з простої метрики **точність (Accuracy)**. Саме дана метрика визначає, яка частка з усіх передбачень моделі є правильно класифікованими, а яка не відповідає дійсності [12]. Ця метрика є важливою, але може бути непродуктивною в ситуаціях, коли класи даних не збалансовані між собою. Наприклад, якщо один клас зображень зустрічається частіше, ніж інші, модель може вчитися визначати саме цей клас краще, знижуючи при цьому точність для менш представлених класів. Для уникнення такої проблеми необхідно грамотно підбирати дані, які будуть використовувати для навчання.

Далі перейдемо до метрики, що відповідає за **точність виявлення (Detection Accuracy)**, вона вимірює, яку частку об'єктів модель правильно визначила серед усіх об'єктів на зображенні. Це важлива метрика для об'єктних

детекторів, оскільки показує, наскільки добре модель впоралася з виявленням розташування об’єкту на зображенні.

Перейдемо до метрики, яка називається **F1-оцінка (F1-Score)**. Вона представляє гармонічне середнє точності і відновлення. Ця метрика враховує як точність, тобто відсоток правильних передбачень, так і відсоток істинних значень в одному числі. F1-оцінка корисна для вирішення проблеми, коли є дисбаланс класів.

Доволі важливою для нашої роботи є метрика **середнього часу визначення (Average Inference Time)**. Дана метрика показує середній час, необхідний моделі для виконання передбачень на одному зображенні чи кадрі [13]. Вона відіграє ключову роль при застосуванні у реальному часі, де швидкість роботи моделі є ключовим фактором. Чим менше середній час визначення, тим ефективніше модель використовується в реальних умовах. Може вимірятися у мілісекундах, секундах чи інших вимірах часу в залежності від потреб застосування.

На завершення розглянемо **метрику продуктивності для класифікації об'єктів (Performance Metric for Object Classification)**. Вона визначає точність або ефективність моделі у визначенні класу чи категорії, до якої належить об'єкт на зображенні. Метрики класифікації включають точність (accuracy), F1-оцінку, матрицю помилок та інші [14], які визначають, наскільки правильно модель класифікує об'єкти на зображеннях.

Кожна з цих метрик надає важливу інформацію про ефективність моделі детектора об'єктів з різних кутів. Точність виявлення вказує на правильність класифікації об'єктів, тоді як середній час визначення та F1-оцінка вказують на швидкість та здатність моделі вирішувати проблему дисбалансу класів.

Для вирішення нашої задачі ключовими будуть метрики Accuracy, F1-Score та Average Inference Time, так як перша та друга вказують на точність класифікації об'єктів на зображеннях, а третя на швидкість виконання передбачень у реальному часі. Для нас це ключові параметри.

3. РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ НА МІСЦЕВОСТІ

Для програмної реалізації було обрано мову програмування Python. Цього року Python посилив свої позиції, як приклад він зайняв №1 у загальному рейтингу «Spectrum» (рис. 3.1) і продовжує збільшувати своє лідерство [15]. Це відбувається загалом тому, що вона стала універсальною і надає величезні можливості у сфері штучного інтелекту.

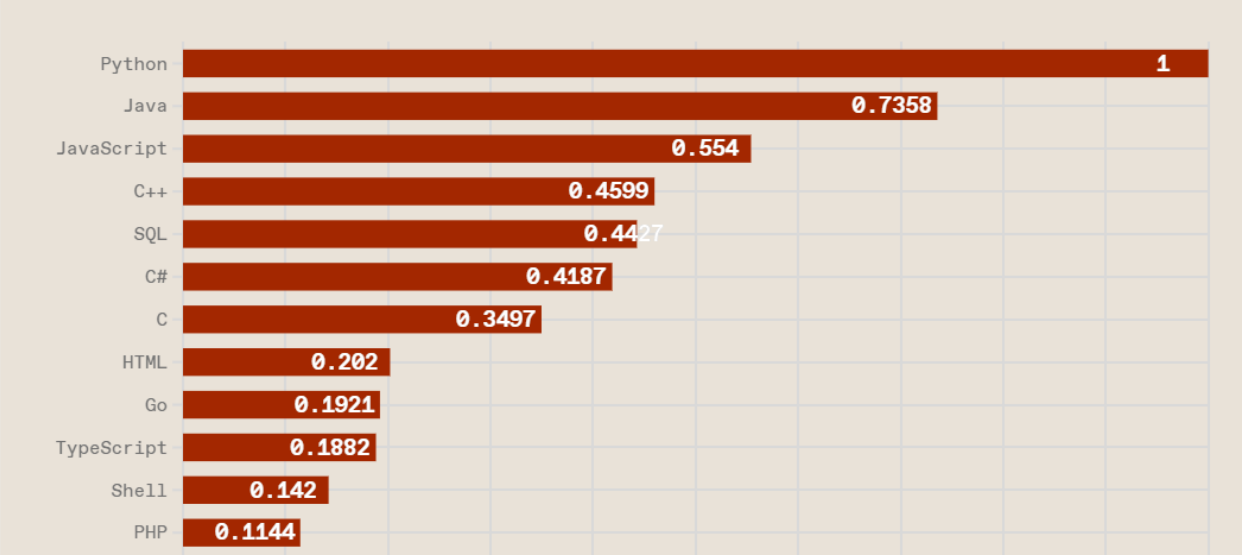


Рисунок 3.1 – Рейтинг мов програмування, з платформи IEEE

На сьогоднішній день Python і розпізнавання зображень є спорідненими поняттями. Python є мовою високого рівня, яка підтримує функціональні, процедурні та об’єктно-орієнтовані стилі програмування, при цьому маючи простий синтаксис і універсальність: цю мову можна використовувати на усіх основних платформах, таких як Windows, MacOS, Linux та UNIX.

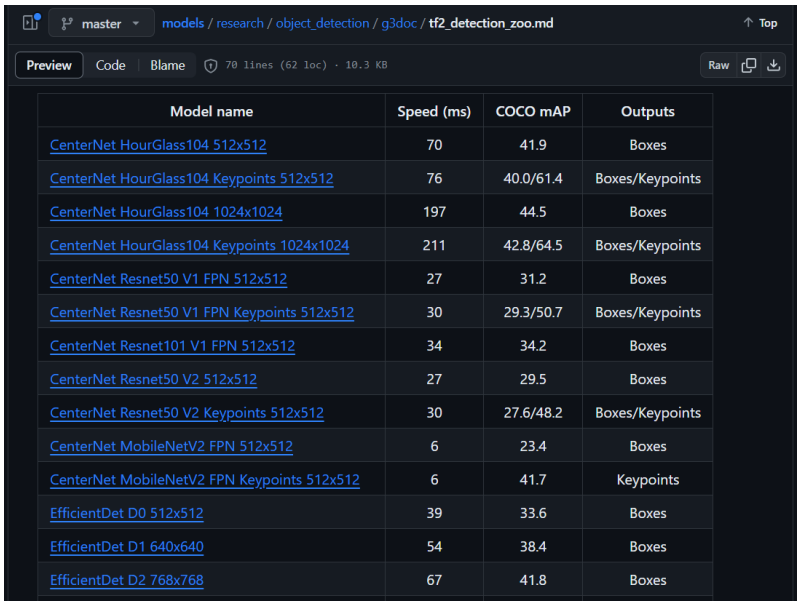
Мова програмування Python надає широкі можливості, для роботи з нейронними мережами, ідентифікацією зображень та рухів [16]. Її сумісність із рядом бібліотек, набором ферймворків та з відкритим кодом TensorFlow, дає розробникам Python зручні інструменти для створення складних алгоритмів. Завдяки цій сумісності використання Python полегшує розробку складних моделей розпізнавання зображень.

Основними перевагами є портативність, велика колекція бібліотек і інструментів, вона є легшою та швидшою за основні мови конкуренти Java та C++ [17].

Тож спираючись на мету нашої роботи та поставлену задачу можна зробити висновок, що дана мова є оптимальною для використання. Але мова то є лише основа, не менш важливо визначитись з головними інструментами. У нашому випадку для навчання моделей було обрано платформу з відкритим кодом, а саме TensorFlow.

TensorFlow Object Detection API є платформою комп'ютерного бачення з відкритим кодом для створення моделей виявлення об'єктів і сегментації зображень [18], які можуть виявити декілька об'єктів на одному зображенні. Більш новішою є версія TensorFlow 2, нею і варто користуватися, бо вона містить нові архітектури.

TensorFlow 2 Object Detection API підтримує такі архітектури і моделі, як CenterNet, EfficientDet, SSD MobileNet, SSD ResNet, Faster R-CNN, ExtremeNet та Mask RCNN. Ці моделі можна завантажити з репозиторію (рис. 3.2) під назвою TensorFlow 2 Detection Model Zoo [19], з якого можна завантажити відповідні файли конфігурації, щоб навчити одну з моделей виявлення об'єктів з нуля, чи скористатися вже попередньо навченою моделлю.



Model name	Speed (ms)	COCO mAP	Outputs
CenterNet HourGlass104 512x512	70	41.9	Boxes
CenterNet HourGlass104 Keypoints 512x512	76	40.0/61.4	Boxes/Keypoints
CenterNet HourGlass104 1024x1024	197	44.5	Boxes
CenterNet HourGlass104 Keypoints 1024x1024	211	42.8/64.5	Boxes/Keypoints
CenterNet Resnet50 V1 FPN 512x512	27	31.2	Boxes
CenterNet Resnet50 V1 FPN Keypoints 512x512	30	29.3/50.7	Boxes/Keypoints
CenterNet Resnet101 V1 FPN 512x512	34	34.2	Boxes
CenterNet Resnet50 V2 512x512	27	29.5	Boxes
CenterNet Resnet50 V2 Keypoints 512x512	30	27.6/48.2	Boxes/Keypoints
CenterNet MobileNetV2 FPN 512x512	6	23.4	Boxes
CenterNet MobileNetV2 FPN Keypoints 512x512	6	41.7	Keypoints
EfficientDet D0 512x512	39	33.6	Boxes
EfficientDet D1 640x640	54	38.4	Boxes
EfficientDet D2 768x768	67	41.8	Boxes

Рисунок 3.2 – Список підготованих моделей

Розглянемо детально основні переваги даного API. Почнемо з того, що дане API просте у використанні, бо забезпечує доступну у застосуванні структуру для реалізації різних моделей виявлення об’єктів, зменшуючи складність створення таких моделей з нуля. Його модульна структура дозволяє користувачам швидко налаштовувати, навчати та розгортати моделі.

Також API пропонує гнучкість у виборі попередньо навчених моделей, зокрема SSD (Single Shot Multibox Detector), Faster R-CNN (Region-based Convolutional Neural Networks) та багатьох інших. Це дозволяє розробникам точно налаштовувати моделі для конкретних випадків використання або наборів даних.

Слід також зазначити, що TensorFlow 2 Object Detection API містить високопродуктивні реалізації найсучасніших моделей виявлення об’єктів. Ці моделі оптимізовані для досягнення чудової точності та ефективності виявлення об’єктів на зображеннях.

Користуючись попередньо підготовленими моделями та передачею навчання, розробники можуть адаптувати ці моделі до нових наборів даних із відносно невеликими обсягами анотованих даних [18]. Ця можливість значно скорочує час і ресурси, необхідні для навчання.

Для початківців важливим плюсом є те, що це API широко забезпечена документацію та має велику кількість прикладів, що полегшує початок роботи. Крім того, будучи частиною екосистеми TensorFlow, він отримує переваги від великої спільноти, що забезпечує постійну підтримку.

Отже зважаючи на свої переваги, TensorFlow 2 Object Detection API спрощує процес розробки, навчання та розгортання моделей виявлення об’єктів, що робить його логічним вибором для цієї роботи.

3.1 Формування навчальних та тестових даних

Для формування навчальних та тестових даних візьмемо готовий набір даних, з якого виберемо частину даних, що підходять для використання у роботі наших моделей. Оригінальний набір даних представлений у форматі Pascal VOC. Формат Pascal VOC (Visual Object Classes) є одним із раніше створених еталонів класифікації та виявлення об’єктів, який надає стандартизований набір даних зображення для розпізнавання класу об’єктів. Формат даних експорту базується на XML і широко застосовується в задачах комп’ютерного зору. Тож в нашому випадку даний формат є зручним тим, що його легко можна змінювати, через простий вид представлення та потім за необхідності доволі просто перевести у інший більш зручний формат.

Після відбору даних вони всі були розділені на три частини: train, valid та test. Тобто на набір для самого навчання, для валідації моделі та для проведення тестів.

Для побудови моделі для виявлення об’єктів наш набір даних було приведено до потрібного вигляду. А саме для початку усі наявні класи були приведені до одного. Адже задачею цієї моделі є саме виявлення, а не ідентифікація об’єкта по його класу. Перейменуємо всі назви класів на ‘object_of_interest’, для цього напишемо функцію, створену для цієї задачі (рис. 3.3) та скористаємося нею (рис. 3.4).

```
def change_class_names(xml_folder):
    for filename in os.listdir(xml_folder):
        if filename.endswith('.xml'):
            xml_path = os.path.join(xml_folder, filename)
            tree = ET.parse(xml_path)
            root = tree.getroot()

            for obj in root.findall('object'):
                name = obj.find('name')
                if name is not None:
                    name.text = 'object_of_interest'

            tree.write(xml_path)
```

Рисунок 3.3 – Функція для зміни класу

```python
folder_path_train = 'dataNext/images/train'
folder_path_test = 'dataNext/images/test'
folder_path_valid = 'dataNext/images/valid'

change_class_names(folder_path_train)
change_class_names(folder_path_test)
change_class_names(folder_path_valid)
```

Python

Рисунок 3.4 – Виклик функції на наборах даних

У результаті роботи функції на нашому наборі даних були перейменовані всі класи, приклад перейменування представлений на рис. 3.5.

```xml
<annotation>
    <folder />
    <filename>-_-_-_-_-21_-_-_-_-_-_-_jpeg_jpg.rf.f18c3c404015b9d68d69d683afe33f42.jpg</filename>
    <path>-_-_-_-_-21_-_-_-_-_-_-_jpeg_jpg.rf.f18c3c404015b9d68d69d683afe33f42.jpg</path>
    <source>
        <database>roboflow.ai</database>
    </source>
    <size>
        <width>640</width>
        <height>640</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>object_of_interest</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <occluded>0</occluded>
        <bndbox>
            <xmin>19</xmin>
            <xmax>585</xmax>
            <ymin>15</ymin>
            <ymax>641</ymax>
        </bndbox>
    </object>
</annotation>
```

Рисунок 3.5 – Приклад зміненої назви класу

Далі дані для тренування моделі детектору об’єктів варто перевести у формат TFRecord. Формат TFRecord є простим форматом для зберігання послідовності двійкових записів, який використовується в TensorFlow для ефективного зберігання великих обсягів даних і керування ними [20]. Він спеціально розроблений для обробки даних TensorFlow і зазвичай використовується для машинного навчання моделей.

Використання файлів TFRecord зазвичай може пришвидшити навчання моделей шляхом оптимізації процесів зчитування даних, оскільки це забезпечує кращу передачу, перемішування та розпаралелювання даних.

Тож вибір формату TFRecord є доволі логічним, але перед початком конвертації необхідно зробити файл (рис. 3.6) зі списком класів для нашого набору даних.

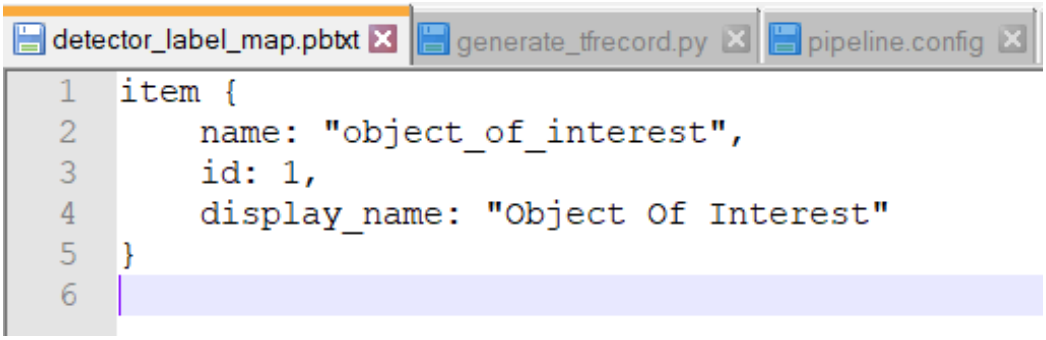


Рисунок 3.6 – Вміст файлу з назвами класів

Далі для конвертації даних з формату Pascal VOC до формату TFRecord скористаємося спеціальним конвертором який надає TensorFlow 2 Object Detection API, приклад його використання наведений на рис. 3.7.

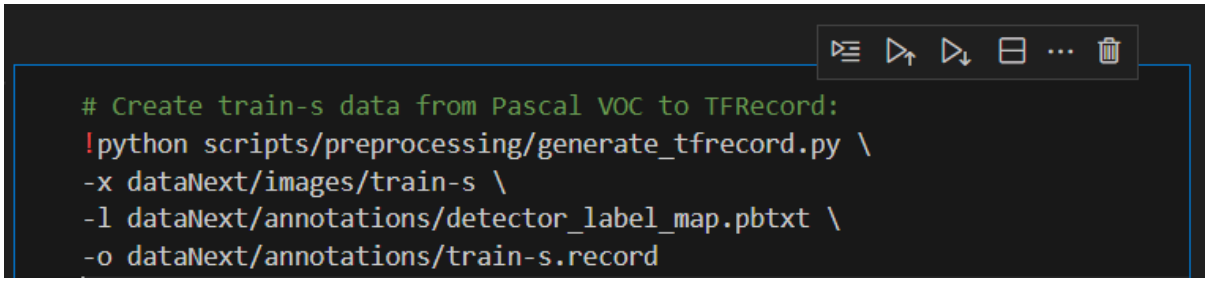


Рисунок 3.7 – Виклик конвертора даних

Після проведення конвертації, набір даних для навчання моделі детектора представлені на рис. 3.8.

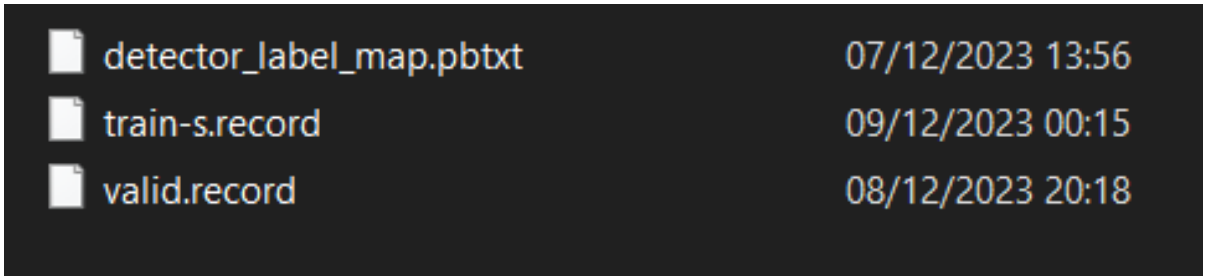


Рисунок 3.8 – Директорія з набором даних для моделі детектора

Далі, використовуючи ті ж методи перетворення, підготуємо дані для навчання моделі класифікатора. Для цього потрібно передивитись увесь тестовий набір даних, який будемо використовувати для навчання класифікатора, та проставити для кожного виділеного об’єкту на зображенні відповідний до нього клас, приклад файлу розмітки з двома класами наведений на рис. 3.9.

```
<annotation>
    <folder />
    <filename>ElZpBp5WoAAXmwf_jpg.rf.d20059baad12687d6dbede1bee816f6f.jpg</filename>
    <path>ElZpBp5WoAAXmwf_jpg.rf.d20059baad12687d6dbede1bee816f6f.jpg</path>
    <source>
        <database>roboflow.ai</database>
    </source>
    <size>
        <width>640</width>
        <height>640</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>Tank</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <occluded>0</occluded>
        <bndbox>
            <xmin>141</xmin>
            <xmax>609</xmax>
            <ymin>31</ymin>
            <ymax>601</ymax>
        </bndbox>
    </object>
    <object>
        <name>Vehicle</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <occluded>0</occluded>
        <bndbox>
            <xmin>73</xmin>
            <xmax>253</xmax>
            <ymin>300</ymin>
            <ymax>464</ymax>
        </bndbox>
    </object>
</annotation>
```

Рисунок 3.9 – Файл розмітки з декількома класами

Після цього приготуємо конфігураційний файл (рис. 3.10), та проведемо процедуру аналогічну перетворенню, а саме скористаємося спеціальним парсером який надає TensorFlow 2 Object Detection API, для конвертації

формату Pascal VOC у формат TFRecord. Тож дані для навчання моделі класифікатора готові і представлені на рис. 3.11.

classifier_label_map_t2.pbtxt | pipeline.config | pipeline.config

```
item {
    name: "Vehicle",
    id: 1,
    display_name: "Vehicle"
}

item {
    name: "Tank",
    id: 2,
    display_name: "Tank"
}

```

Рисунок 3.10 – Список класів для нашого набору даних

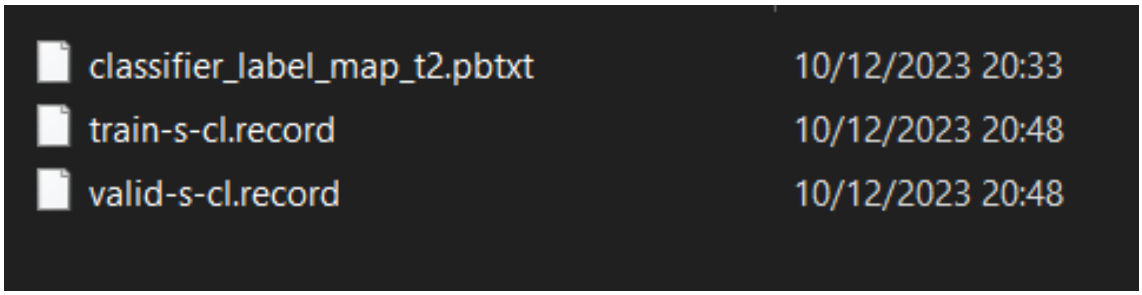


Рисунок 3.11 – Директорія з набором даних для моделі класифікатора

Тестовий набір даних необхідно також передивитися та проставити відповіді класи для кожного об’єкту на зображенні, але на відміну від навчального набору даних, тестові дані залишимо у форматі Pascal VOC та використаємо їх потім для тестування та створення додаткових наборів тестових даних.

Ще підготуємо тестові дані для того, щоб провести оцінку якості моделі класифікатора, для цього візьмемо підготовлений тестовий набір даних та виріжемо з зображень цього набору всі об’єкти, котрі відмічені у файлах розмітки, для кожного вирізаного об’єкту створимо простий xml файл, котрий буде зберігати ідентифікатор класу, до якого відноситься об’єкт на його назву, після цього збережемо зображення вирізаного об’єкту у відповідний йому xml

файл. Для проведення такої процедури скористаємося написаною функцією crop_objects_from_images. Її можна переглянути детально в Додатку А. Виклик даної функції представлений на рис. 3.12

```
# crop objects from images
cl_test_folder_path = "dataNext/images/test-s-cln"
cl_crop_folder_path = "dataNext/images/test-s-cl-crop"
crop_objects_from_images(cl_test_folder_path, cl_crop_folder_path)
✓ 1.1s

COMPLETE
```

Рисунок 3.12 – Виклик функції crop_objects_from_images

В результаті усі тестові набори представлені у чотирьох директоріях: у папці test - оригінальні не відібрані дані, у папці test-s-cl - відібрані дані для використання в тестуванні роботи системи, у папці test-s-cln - дані відібрані та розділені на класи та у папці test-s-cl-crop - відібрані та підготовлені дані для оцінки роботи класифікатора. представлені на рис.3.13.

Рисунок 3.13 – Директорія з усіма тестовими даними

Отже підготовка вхідних даних для навчання та тестування нашої інформаційної технології розпізнавання об’єктів на зображенні в умовах обмежених ресурсів виконана.

3.2 Короткий опис програмного забезпечення

Програма розроблена на мові Python з використанням широкого набору бібліотек, у середовищі Jupyter Notebook, оскільки воно є зручним для запуску окремих частин коду на мові Python. Для навчання моделей було використано

TensorFlow 2 Object Detection API, через його зручніть та простоту застосування.

Для початку було проведено навчання моделі SSD MobileNet Lite у якості детектора об’єктів на зображеннях. Для цього була завантажена заздалегідь навчена та підготовлена модель, її використання необхідне для того, щоб пришвидшити навчання на нашому наборі даних. Для навчання моделі був створений та відредагований конфігураційний файл (рис.3.14), в якому були зазначені шляхи до заздалегідь навченої та підготовленої моделі, файлів мапінгу класів та до тренувальних даних.

```
  fine_tune_checkpoint: "dataNext/preTrainedModels/ssd_mobilenet_v2_lite/checkpoint/ckpt-0" #\/
  num_steps: 6200
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  fine_tune_checkpoint_version: V2
}
train_input_reader {
  label_map_path: "dataNext/annotations/detector_label_map.pbtxt" #\/ Path to label map file
  tf_record_input_reader {
    input_path: "dataNext/annotations/train-s.record" #\/ Path to training TFRecord file
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}
eval_input_reader {
  label_map_path: "dataNext/annotations/detector_label_map.pbtxt" #\/ Path to label map file
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "dataNext/annotations/valid.record" #\/ Path to valid TFRecord
  }
```

Рисунок 3.14 – Конфігураційний файл для моделі детектора

Далі, користуючись функціоналом TensorFlow 2 Object Detection API, запустимо навчання нашого детектору об’єктів (рис.3.15), для того, щоб на навчальному наборі даних модель навчилась знаходити потрібні об’єкти.

```
#detector
!python model_main_tf2.py \
    --model_dir=dataNext/models/ssd_mobilenet_v2_lite \
    --pipeline_config_path=dataNext/models/ssd_mobilenet_v2_lite/pipeline.config
```

Рисунок 3.15 – Навчання моделі детектора об’єктів

Після проведення навчання, директорія з вже навченою моделлю детектора містить в собі дані результату навчання, конфігураційний файл моделі та журнал ходу навчання (рис. 3.16).

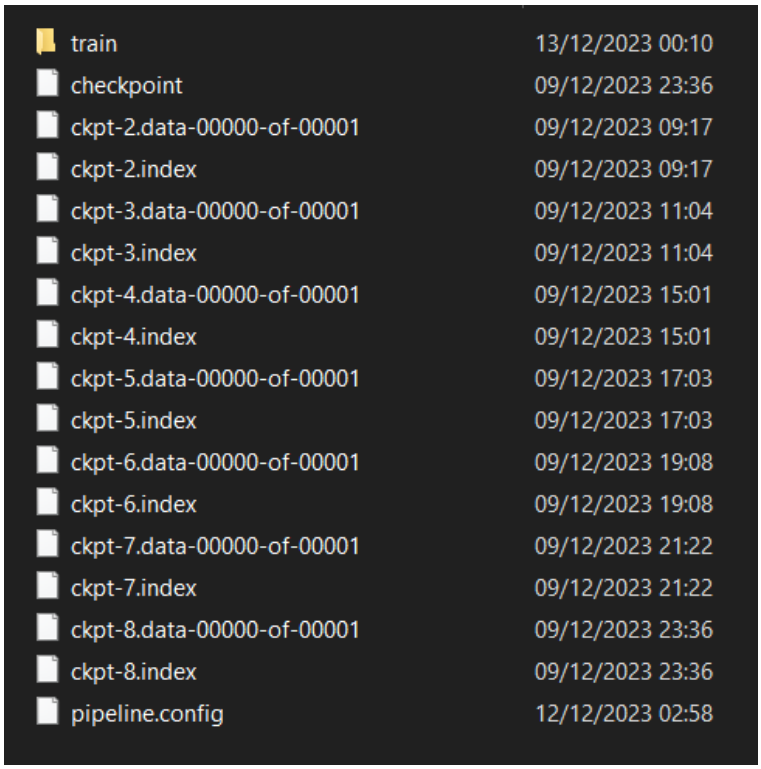


Рисунок 3.16 – Директорія з навченою моделлю

У ході навчання було здійснено 6200 кроків, на графіках, представлених на рис.3.17 - 3.20 можна побачити, що модель детектора навчилася доволі ефективно розпізнавати об’єкти.

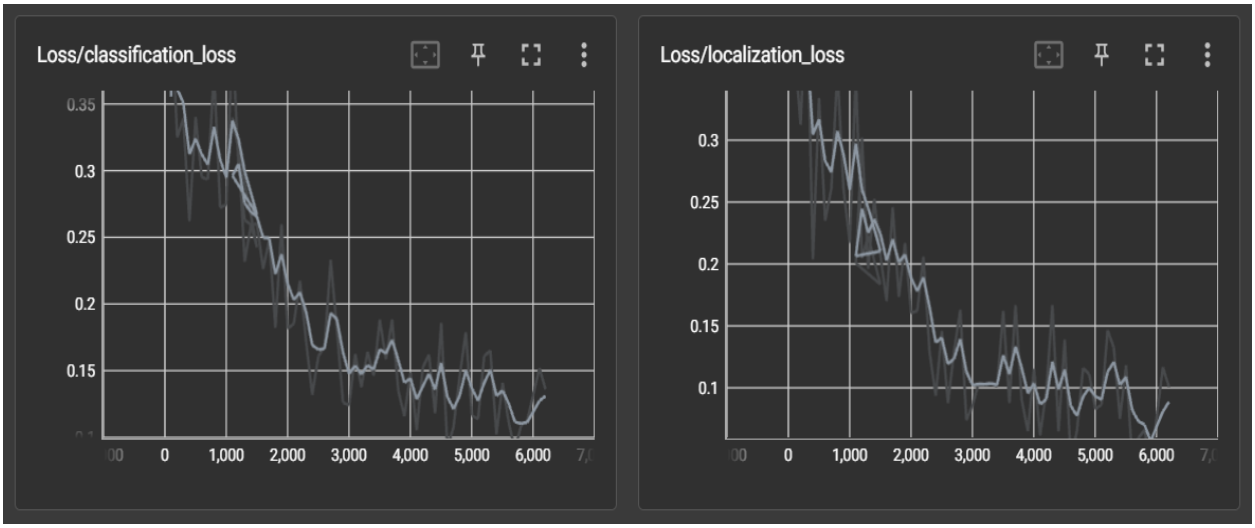


Рисунок 3.17 – Результати навчання моделі детектора, частина 1

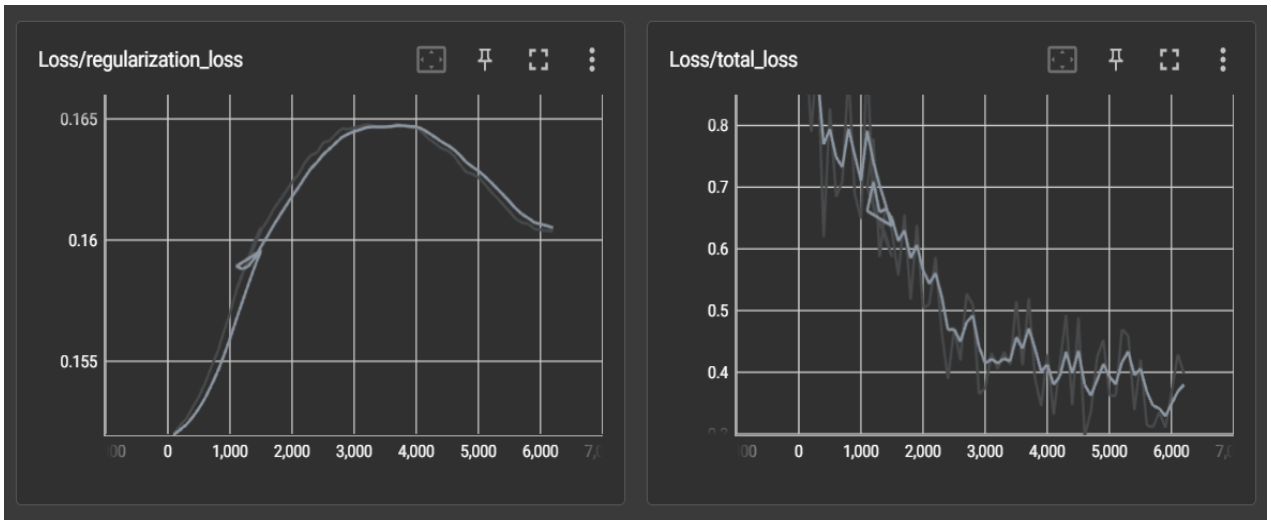


Рисунок 3.18 – Результати навчання моделі детектора, частина 2

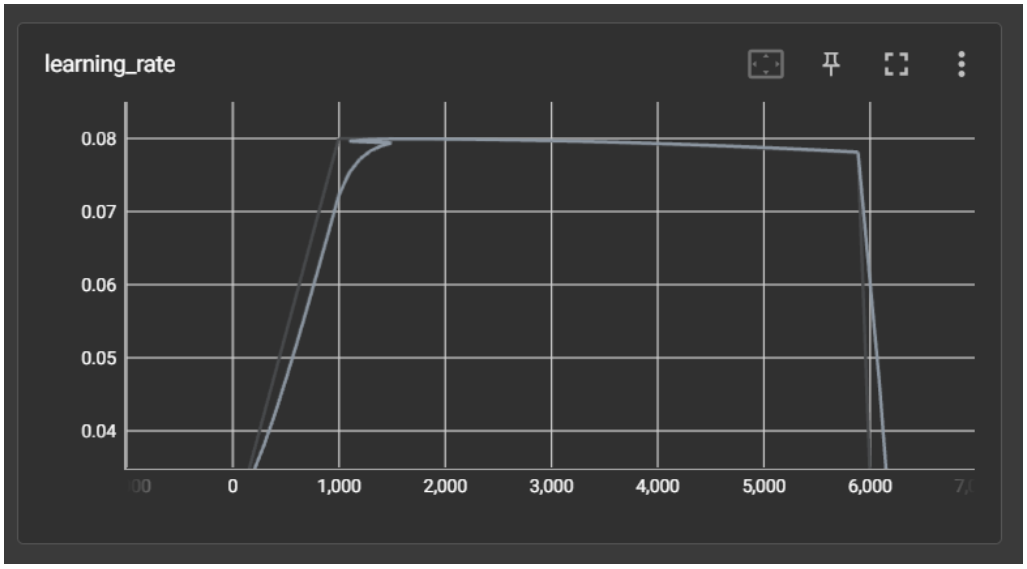


Рисунок 3.19 – Результати навчання моделі детектора, частина 3

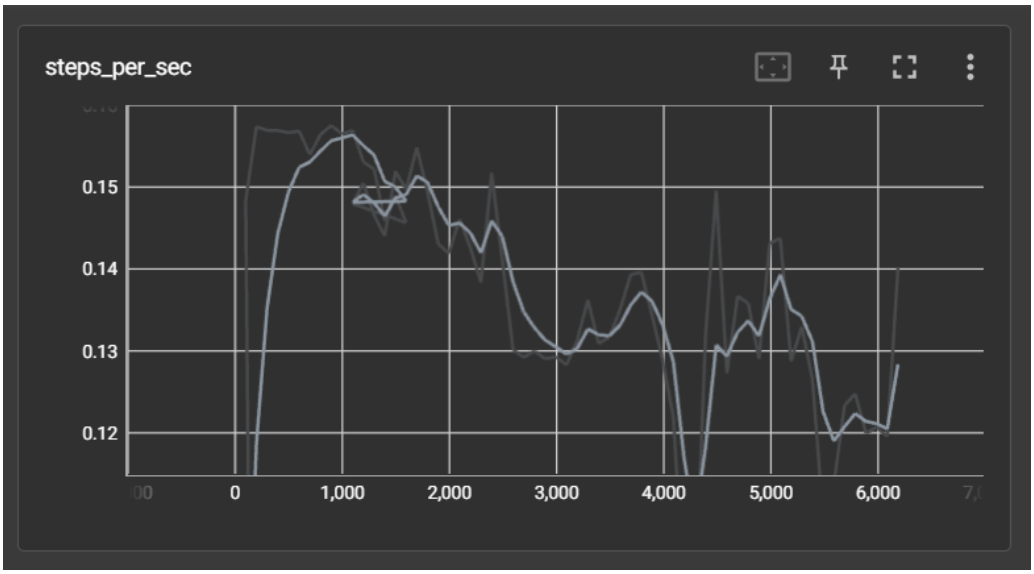


Рисунок 3.20 – Результати навчання моделі детектора, частина 4

Але все ж для досягнення кращих результатів потрібно було б продовжити навчання та для даної роботи цього є достатньо.

Далі проведемо навчання моделі класифікації об’єктів на зображенні. Для цього, візьмемо заздалегідь навчену модель, щоб пришвидшити навчання на нашому наборі даних та створимо конфігураційний файл (рис.3.21) по аналогії до конфігураційного файлу детектора, в якому були зазначені шляхи до тренувальних даних.

```
  num_steps: 3500
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  fine_tune_checkpoint_version: V2
}
train_input_reader {
  label_map_path: "dataNext/annotations/classifier_label_map_t2.pbtxt"
  tf_record_input_reader {
    input_path: "dataNext/annotations/train-s-cl.record"
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}
eval_input_reader {
  label_map_path: "dataNext/annotations/classifier_label_map_t2.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "dataNext/annotations/valid-s-cl.record"
```

Рисунок 3.21 – Конфігураційний файл для моделі класифікатора

Після цього по аналогії до детектора навчимо і класифікатор, використовуючи model_main_tf2.py запустимо навчання нашого класифікатора об’єктів (рис. 3.22).

```
#clasifier 3
!python model_main_tf2.py \
    --model_dir=dataNext/models/classifier_ssd_mobilenet \
    --pipeline_config_path=dataNext/models/classifier_ssd_mobilenet/cl_pipeline.config \
    --checkpoint_every_n=200
```

Рисунок 3.22 – Навчання моделі класифікатора об’єктів

Після проведення навчання, директорія з вже навченою моделлю класифікатора містить в собі дані результату навчання, конфігураційний файл моделі та журнал ходу навчання (рис. 3.23).

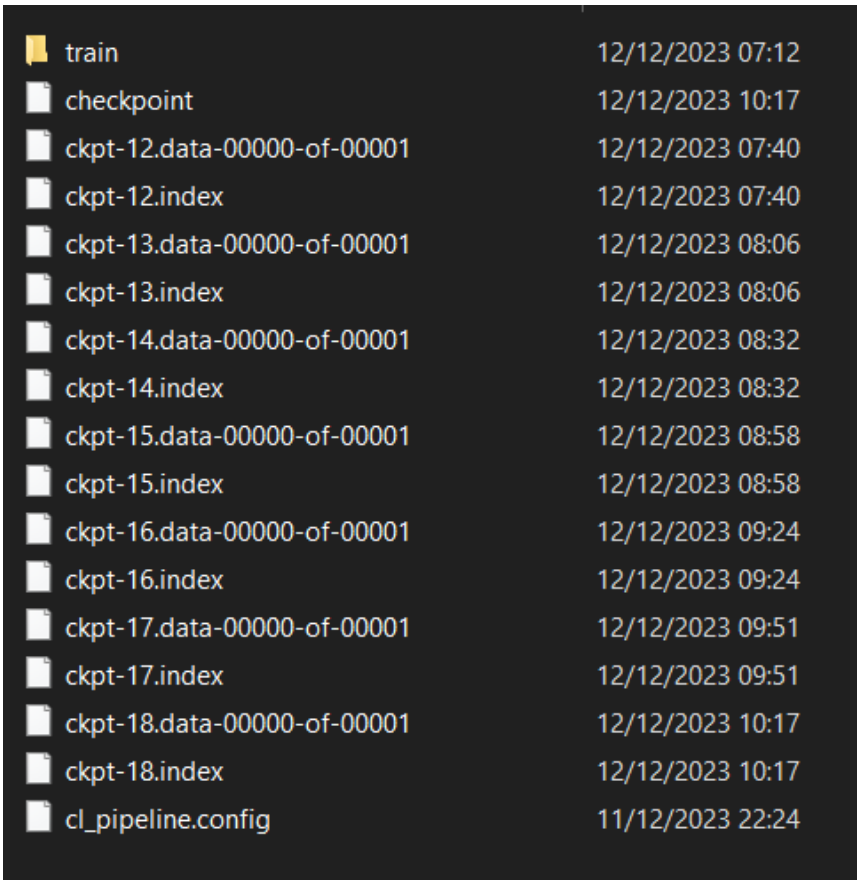


Рисунок 3.23 – Директорія з навченою моделлю

У ході навчання класифікатора було здійснено 3500 кроків, на графіках, представлених на рис. 3.24 - 3.27 можна побачити, що модель класифікатора навчилась розпізнавати об’єкти на достатньому рівні.

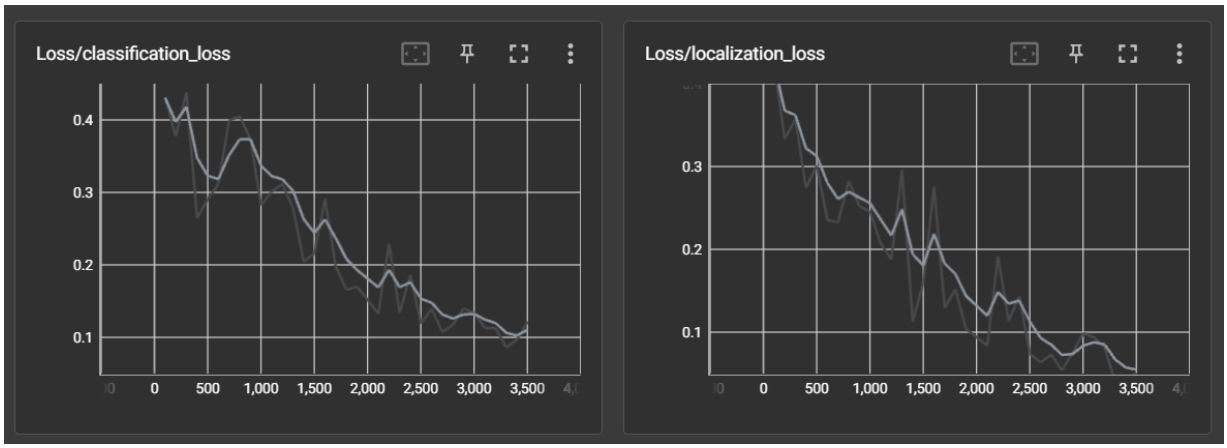


Рисунок 3.24 – Результати навчання моделі детектора, частина 1

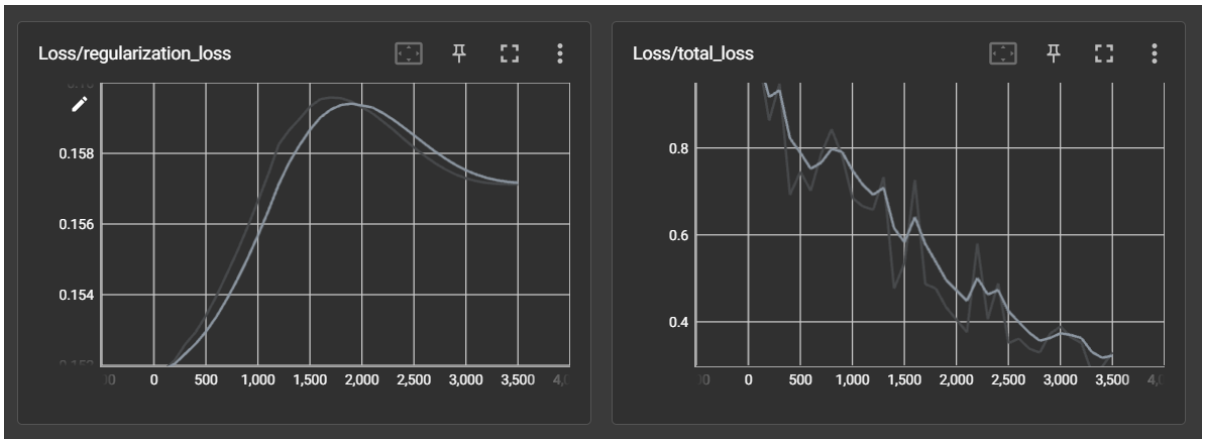


Рисунок 3.25 – Результати навчання моделі детектора, частина 2

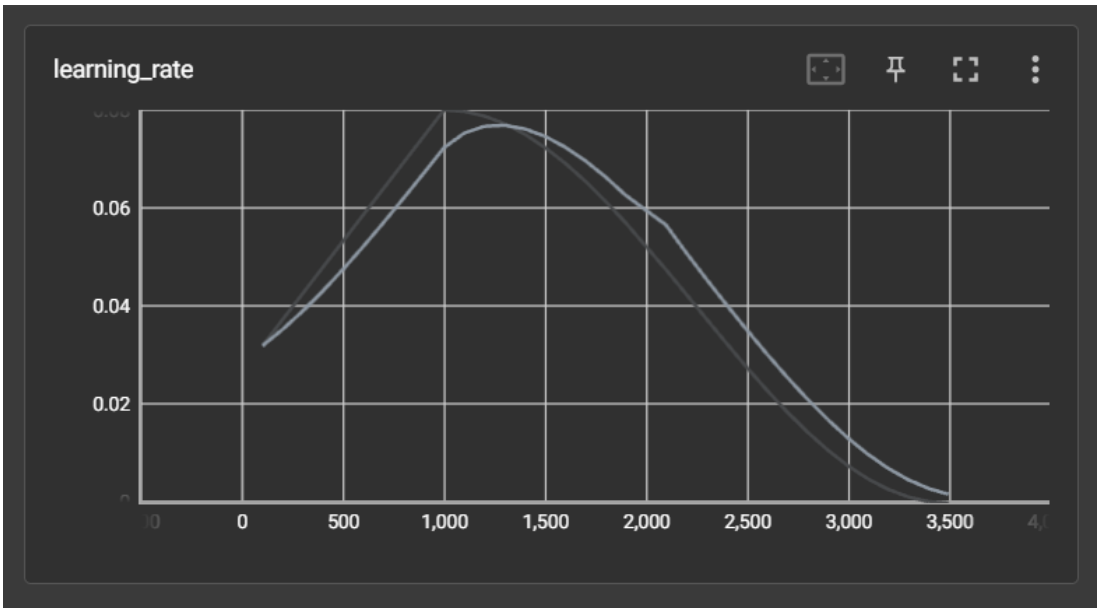


Рисунок 3.26 – Результати навчання моделі детектора, частина 3

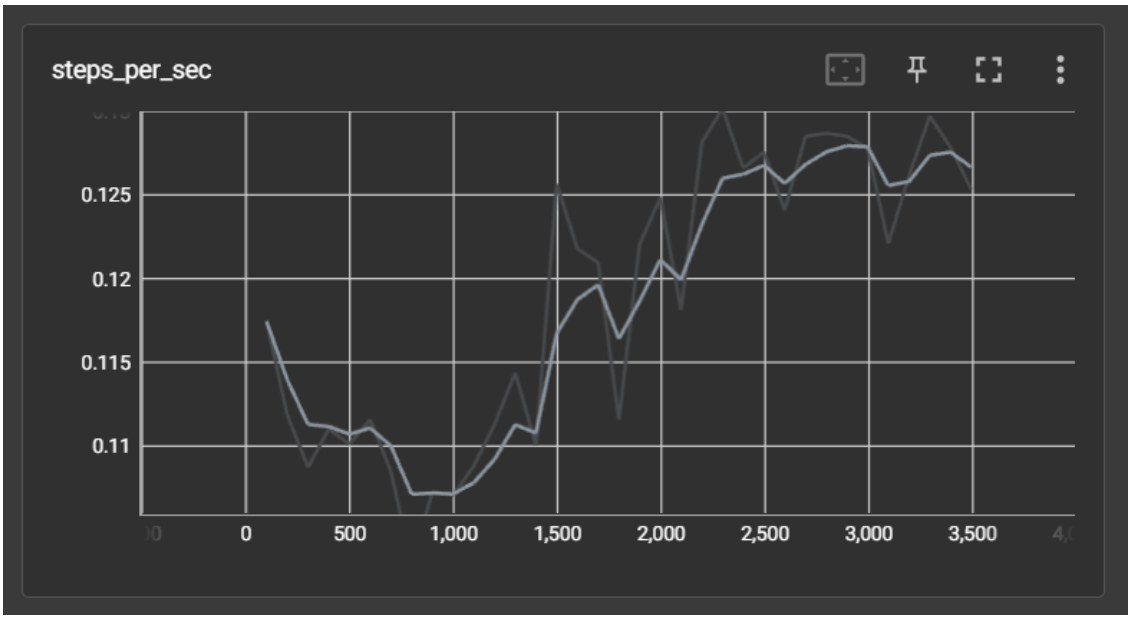


Рисунок 3.27 – Результати навчання моделі детектора, частина 4

Далі після завершення навчання збережемо наші моделі, скориставшись інструментом exporter_main_v2.py, який є вбудованим у TensorFlow 2 Object Detection API та є дуже зручним для отримання навченої моделі у форматі, який надає можливість її завантажувати та використовувати, команди на виконання експорту для моделей детектора та класифікатора, представлені на рис. 3.28 та рис. 3.29.

```
#detector
!python exporter_main_v2.py \
    --input_type image_tensor \
    --pipeline_config_path dataNext/models/ssd_mobilenet_v2_lite/pipeline.config \
    --trained_checkpoint_dir dataNext/models/ssd_mobilenet_v2_lite/ \
    --output_directory dataNext/exported-models/lite_model
```

Рисунок 3.28 – Експорт моделі детектора

```
#classifier
!python exporter_main_v2.py \
    --input_type image_tensor \
    --pipeline_config_path dataNext/models/classifier_ssd_mobilenet/cl_pipeline.config \
    --trained_checkpoint_dir dataNext/models/classifier_ssd_mobilenet/ \
    --output_directory dataNext/exported-models/classifier_model_mobilenet
```

Рисунок 3.29 – Експорт моделі класифікатора

Після закінчення навчання наших моделей вже можна перейходити до опису самої програмної реалізації, перед запуском будь яких компонентів програми має підключити цілий ряд бібліотек, які допомагають в роботі з зображеннями (рис. 3.30).

```
import tensorflow as tf
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
import cv2
import numpy as np
import os
import time
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import xml.etree.ElementTree as ET
✓ 15.7s
```

Рисунок 3.30 – Підключення бібліотек

Перед проведенням тестів на створеній моделі необхідно провести ініціалізацію головних параметрів (рис. 3.31). За необхідності треба змінити

шляхи до місць зберігання моделей, файлів мапінгу класів та тестового набору зображень.

```python
#init
d_model_dir = 'dataNext/exported-models/lite_model/saved_model/'
c_model_dir = 'dataNext/exported-models/classifier_model_mobilenet/saved_model/'
label_map_path = 'dataNext/annotations/detector_label_map.pbtxt'
c_label_map_path = 'dataNext/annotations/classifier_label_map_t2.pbtxt'
images_folder = 'dataNext/images/test-s-cl'

#detector
d_model = tf.saved_model.load(d_model_dir)
detection_model = d_model.signatures['serving_default']
label_map = label_map_util.load_labelmap(label_map_path)
categories = label_map_util.convert_label_map_to_categories(
    label_map,
    max_num_classes=1,
    use_display_name=True
)
category_index = label_map_util.create_category_index(categories)

#clasifier
c_model = tf.saved_model.load(c_model_dir)
classifier_model = c_model.signatures['serving_default']
c_label_map = label_map_util.load_labelmap(c_label_map_path)
c_categories = label_map_util.convert_label_map_to_categories(
    c_label_map,
    max_num_classes=2,
    use_display_name=True
)
c_category_index = label_map_util.create_category_index(c_categories)
```

✓ 18.4s Python

Рисунок 3.31 – Ініціалізація головних параметрів програми

Після цього вже можна переходити то проведення виявлення та класифікації об’єктів на зображенні. Для цього треба використовувати функцію main_detector, код якої можна знайти у Додатку А. Ця функція є головною в нашій системі, адже вона проводить виявлення об’єктів на тестовому наборі даних, використовуючи модель детектора, вирізає виявлені об’єкти, та передає їх на класифікацію моделі, яка відповідає за уточнення класу. Після виявлення та класифікації об’єкта на зображенні при умові досягнення достатньої точності, даний об’єкт виділяється рамкою на зображенні та підписується назвою класу та показником точності. Усі проаналізовані зображення з визначеними на них об’єктами зберігаються у директорію, яка передається цій функції, як параметр. Функція повертає

значення, необхідні для визначення середньої швидкості обробки одного зображення детектором та класифікатором. Діаграма на якій представлена робота даної функції у спрощеному вигляді наведена на рис. 3.32.

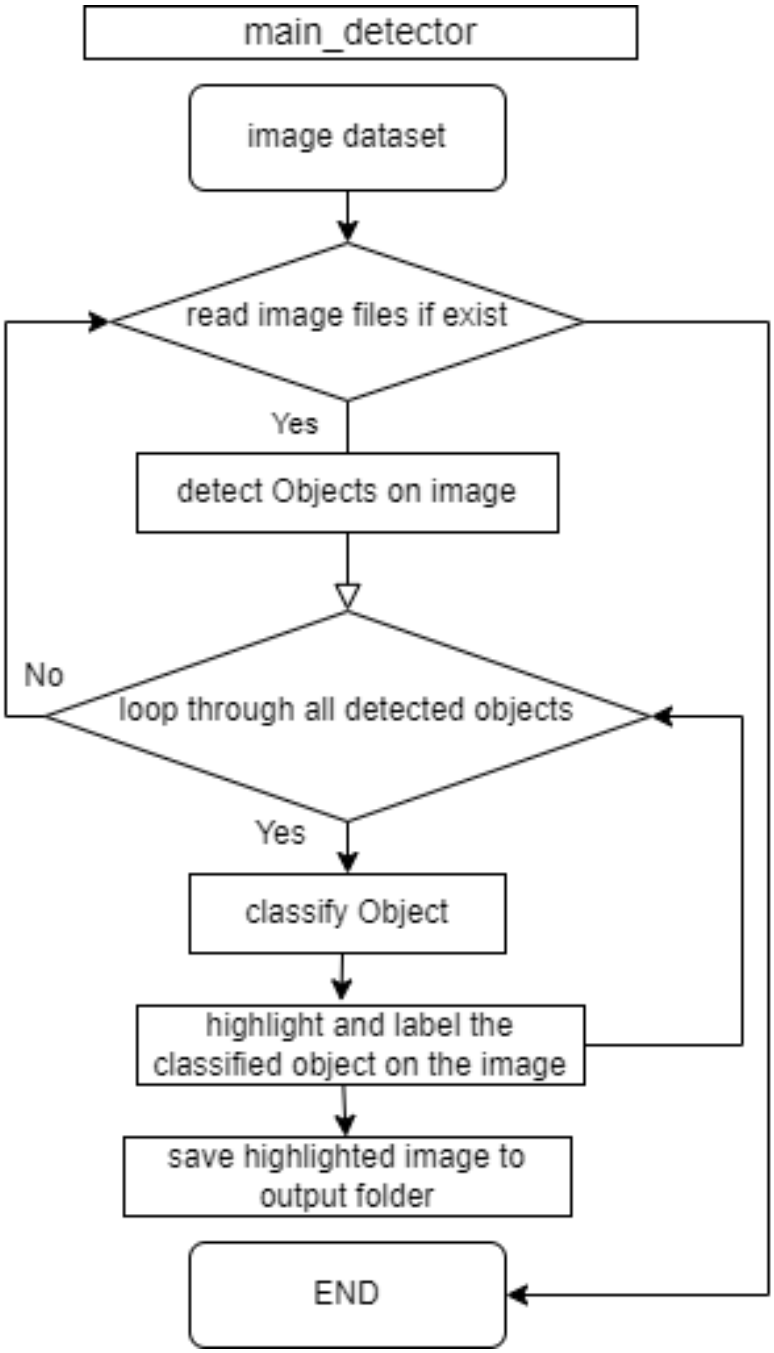


Рисунок 3.32 – Діаграма функції main_detector

Приклад виклику даної функції наведений в наступному розділі, як і результати розпізнавання об’єктів на зображенні в умовах обмежених ресурсів.

3.3 Постановка та аналіз результатів експериментів

В даному розділі ми розглянемо результати роботи нашої системи на сформованому тестовому наборі даних. Також отримаємо та проаналізуємо оцінки метрик, які формує наша система. Почнемо з виклику нашої головної функції main_detector (рис. 3.33), яка і починає процес обробки тестового набору даних.

```
output_folder = 'dataNext/images/test-res'
total_detection_time, total_classification_time,num_images = main_detector(output_folder)
```

✓ 1m 3.2s Python

Рисунок 3.33 – Виклик функції main_detector

Після закінчення виконання головної функції наведемо декілька типових прикладів результатів виявлення об’єктів на зображенні. Вони представлені на рис. 3.34 - 3.39.

Рисунок 3.34 – Результат виявлення, перший приклад

Рисунок 3.35 – Результат виявлення, другий приклад

Рисунок 3.36 – Результат виявлення, третій приклад

Рисунок 3.37 – Результат виявлення, четвертий приклад

Рисунок 3.38 – Результат виявлення, п’ятий приклад

Рисунок 3.39 – Результат виявлення, шостий приклад

Як можемо бачити, виявлення об’єктів на зображеннях є більш менш точним, але варто зазначити, що через те, що в навчальному наборі даних була переважна більшість об’єктів класу ‘Vehicle’ то їх система розпізнає краще, ніж об’єкти класу ‘Tank’. Тому для оптимізації системи варто зробити навчальну вибірку більш пропорційною для усіх класів, та на покращеному наборі даних продовжити навчання системи. Також трапляються випадки коли модель детектора знаходить об’єкт, але точній прогнозу моделі класифікатора не є достатньою, щоб вказувати клас для знайденого об’єкту, в таких випадках ми можемо бачити підпис ‘Unknown’.

Після виконання розпізнавання, система повертає дані, що містять загальний час роботи моделі детектора, загальний час роботи моделі класифікатора та кількість оброблених зображень. Ці дані необхідні для обрахування середнього часу визначення (Average Inference Time). Користуючись простою формулою, а саме загальний час визначення моделі

поділений на кількість оброблених зображень отримаємо результати, для обох наших моделей (рис. 3.40).

```
# Calculate average inference times
average_detection_time = total_detection_time / num_images
average_classification_time = total_classification_time / num_images

print(f"Average Detection Time: {average_detection_time:.4f} sec")
print(f"Average Classification Time: {average_classification_time:.4f} sec")
print(f"Average Inference Time: {(average_classification_time + average_detection_time):.4f} sec")
```

✓ 0.0s

```
Average Detection Time: 0.1729 sec
Average Classification Time: 0.2043 sec
Average Inference Time: 0.3772 sec
```

Рисунок 3.40 – Розрахунок Average Inference Time

Як можна помітити детектування об’єктів займає менше часу ніж класифікація знайдених об’єктів. Також згідно результату, загальний час обробки одного зображення обома моделями складає 0.3772 секунди. Що є середнім результатом, для систем даного типу.

На останок розрахуємо метрики точності для моделі класифікатора. Для цього скористаємося створеною функцією classify_and_compare, її код можна подивитися в Додатку А. Задачею даної функції є обробка підготовленого тестового набору ‘test-cl-crop’, який містить зображення з вже вирізаними об’єктами та файли xml формату, які зберігають визначену назву класу об’єкта на зображенні та його ідентифікаційний номер згідно файлу з мапінгом.

Отже дана функція передає набір зображень об’єктів класифікатору та зберігає результат наданого ним передбачення і фактичне значення класу, яке завантажується з xml файлу, результатом роботи функції є два масиви, один з передбаченими класифікатором класами, а інший з фактичними класами, які були проставлені відповідно до зображених об’єктів. Саме ці результати ми і використаємо для обчислення метрик Accuracy та F1-Score для нашої моделі класифікатора. Виклик функції classify_and_compare та результати обчислень метрик Accuracy та F1-Score наведено на рис. 3.41.

```
#classification:
c_model_dir = 'dataNext/exported-models/classifier_model_mobilenet/saved_model/'
cl_crop_folder_path = "dataNext/images/test-s-cl-crop"
model_classified_labels, actual_labels = classify_and_compare(cl_crop_folder_path, c_model_dir)
```

✓ 1m 8.0s

```
#metrics:
accuracy = accuracy_score(actual_labels, model_classified_labels)
f1 = f1_score(actual_labels, model_classified_labels)

print("Accuracy:", accuracy)
print("F1 Score:", f1)
```

✓ 0.0s

```
Accuracy: 0.6304347826086957
F1 Score: 0.7548076923076924
```

Рисунок 3.41 – Результат розрахунку метрик

Як ми можемо помітити з результатів оцінок Accuracy та F1-Score, модель класифікатора потребує подальшого навчання та оптимізації. Також є зрозумілим, що як і було зазначено раніше є проблема з незбалансованою кількістю представників класу ‘Tank’ та ‘Vehicle’ на користь останнього, через це класифікація об’єкту з класу ‘Tank’ є недостатньо точною. Але слід відмітити те, що при достатній кількості навчання та урівноважуванню навчальної вибірки між усіма класами точність має суттєво покращитися, а завдяки використанню у нашій системі легких моделей вона може працювати в умовах обмежених ресурсів. Тож дана система може розвиватися та вдосконалюватися.

ВИСНОВКИ

У ході виконання магістерської кваліфікаційної роботи були розглянуті сучасні тенденції у сфері розпізнавання предметів на зображеннях, моделі та методи, котрі дозволяють зменшити використання ресурсів при незначних втратах у точності розпізнавання об’єктів, та обрані ті котрі є оптимальними для виконання поставленої завдання. Було обрано мову програмування та інструменти котрі є простими та зручними для роботи з зображеннями та моделями розпізнавання.

Були вибрані та приведені до потрібного стану дані, котрі в подальшому були розділені на тренувальні та тестові набори. Отримані набори даних застосували для навчання моделей детектора та класифікатора об’єктів. На основі цих моделей було реалізовано систему розпізнавання об’єктів на зображенні в умовах обмежених ресурсів. Використовуючи тестовий набір даних була проведена перевірка можливостей розробленої системи розпізнавання.

Спираючись на результати розпізнавання об’єктів на тестовому наборі зображень та на результати використаних метрик точності та часу були зроблені висновки, щодо якості та ефективності розробленої інформаційної технології розпізнавання зображень в умовах ресурсних обмежень.

Згідно аналізу результатів роботи даної інформаційної системи можна зробити висновок, що вона є працездатною та виконує функції розпізнавання на достатньому рівні, при використанні легких моделей детектора та класифікатора, що дозволяє зменшити необхідні ресурси. Водночас дана система може бути оптимізована та вдосконалена у подальшому.

СПИСОК ЛІТЕРАТУРИ

1. Ningning Ma, Xiangyu Zhang, Light-weight CNN Architecture Design for Fast Inference, 2018. 3-10 с. URL: https://openaccess.thecvf.com/content_ECCV_2018/papers/Ningning_Light-weight_CNN_Architecture_ECCV_2018_paper.pdf (дата звернення 02.12.2023).
2. Redmon, Joseph, Santosh Divvala, You only look once: Unified, real-time object detection, 2016. URL: https://arxiv.org/abs/1506.02640 (дата звернення 02.12.2023).
3. Ren, Shaoqing, Faster R-CNN: Towards real-time object detection with region proposal networks, 2015. 91-99 с. URL: https://arxiv.org/abs/1506.01497 (дата звернення 02.12.2023).
4. Liu, Wei, SSD-MobileNet: An object detection model for resource-constrained devices, 2018. 290-299 с.
5. Peng Zhou, Bingbing Ni, Scale-Transferrable Object Detection, 2018. 2-4 с. URL: https://openaccess.thecvf.com/content_cvpr_2018/CameraReady/1376.pdf (дата звернення 10.12.2023).
6. Tailin Liang, John Glossner, Pruning and Quantization for Deep Neural Network Acceleration, 2021. URL: https://arxiv.org/abs/2101.09671 (дата звернення 02.12.2023).
7. Maxim Bonnaerens, Matthias Freiberger, Anchor Pruning for Object Detection, 2022. URL: https://arxiv.org/abs/2104.00432 (дата звернення 12.12.2023).
8. Linfeng Zhang, Jiebo Song, Be Your Own Teacher: Improve the Performance of Convolutional Neural Networks via Self Distillation, 2019. URL: https://arxiv.org/abs/1905.08094 (дата звернення 02.12.2023).
9. Vidish Mehta, Object Detection using SSD Mobilenet V2, 2021. URL: https://vidishmehta204.medium.com/object-detection-using-ssd-mobilenet-v2-7ff3543d738d (дата звернення 12.12.2023).

10. Anurag Singh Choudhary, Object Detection Using YOLO And Mobilenet SSD, 2022. URL: https://www.analyticsvidhya.com/blog/2022/09/object-detection-using-yolo-and-mobilenet-ssd/ (дата звернення 12.12.2023).
11. Howard, Andrew G, Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017. 2-8 с. URL: https://arxiv.org/abs/1704.04861 (дата звернення 02.12.2023).
12. Renu Khandelwal, Evaluating performance of an object detection model, 2020. URL: https://towardsdatascience.com/evaluating-performance-of-an-object-detection-model-137a349c517b (дата звернення 10.12.2023).
13. Amnon Geifman, The Correct Way to Measure Inference Time of Deep Neural Networks, 2023. URL: https://deci.ai/blog/measure-inference-time-deep-neural-networks/ (дата звернення 12.12.2023).
14. Sumeet Kumar Agrawal, Metrics to Evaluate your Classification Model to take the right decisions, 2023. URL: https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/ (дата звернення 12.12.2023).
15. The Top Programming Languages 2023: веб-сат. URL: https://spectrum.ieee.org/the-top-programming-languages-2023 (дата звернення 02.12.2023).
16. Cleophas Mulongo, Top Programming Languages For Image Recognition, 2022. URL:https://www.technotification.com/2018/11/programming-for-image-recognition.html (дата звернення 02.12.2023).
17. Marcin Frąckiewicz, AI and Computer Vision: Languages and Libraries for Object Detection and Recognition, 2023. URL: https://ts2.space/en/ai-and-computer-vision-languages-and-libraries-for-object-detection-and-recognition/ (дата звернення 12.12.2023).
18. TensorFlow Object Detection API tutorial: веб-сайт. URL: https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/ (дата звернення 02.12.2023).

19. Derrick Mwiti, Object detection with TensorFlow 2 Object detection API, 2022. URL: https://www.machinelearningnuggets.com/object-detection-with-tensorflow-2-object-detection-api/ (дата звернення 12.12.2023).
20. Pascal Janetzky, Training a Neural Network on TFRecord files: веб-сайт. URL: https://towardsdatascience.com/training-a-neural-network-on-tfrecord-files-8bff3b6e9ff4 (дата звернення 02.12.2023).

ДОДАТОК А

Функція change_class_names

```
def change_class_names(xml_folder):
    for filename in os.listdir(xml_folder):
        if filename.endswith('.xml'):
            xml_path = os.path.join(xml_folder, filename)
            tree = ET.parse(xml_path)
            root = tree.getroot()
            for obj in root.findall('object'):
                name = obj.find('name')
                if name is not None:
                    name.text = 'object_of_interest'
            tree.write(xml_path)
```

Функція distinct_name_fields

```
def distinct_name_fields(xml_folder):
    distinct_names = set()
    for filename in os.listdir(xml_folder):
        if filename.endswith('.xml'):
            xml_path = os.path.join(xml_folder, filename)
            tree = ET.parse(xml_path)
            root = tree.getroot()
            for obj in root.findall('object'):
                name = obj.find('name')
                if name is not None:
                    distinct_names.add(name.text)
    return list(distinct_names)
```

Функція find_highest_score_index

```
def find_highest_score_index(classes, scores, threshold=0.4):
    class_scores = {}
    for idx, class_id in enumerate(classes):
        score = scores[idx]
        if class_id not in class_scores or score > class_scores[class_id]:
            class_scores[class_id] = score
    high_score_index = np.argmax(scores)
    if 2 in class_scores and class_scores[2] >= threshold:
        high_score_index = np.argmax(scores[classes == 2])
    return high_score_index
```

Функція main_detector

```
def main_detector(output_folder):
    os.makedirs(output_folder, exist_ok=True)
    total_detection_time = 0
    total_classification_time = 0
    num_images = 0
    for image_file in os.listdir(images_folder):
        if image_file.endswith('.jpg'):
            image_path = os.path.join(images_folder, image_file)
            image_np = cv2.imread(image_path)
            image_np = cv2.cvtColor(image_np, cv2.COLOR_BGR2RGB)
            start_time = time.time()
            input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0),
dtype=tf.uint8)
            detections = detection_model(input_tensor)
            detection_time = time.time() - start_time
            total_detection_time += detection_time
            boxes = detections['detection_boxes'][0].numpy()
```

```
        classes = detections['detection_classes'][0].numpy().astype(np.int32)
        scores = detections['detection_scores'][0].numpy()
        for i in range(len(scores)):
            if scores[i] > 0.6:
                ymin, xmin, ymax, xmax = boxes[i]
                im_height, im_width, _ = image_np.shape
                (left, right, top, bottom) = (int(xmin * im_width), int(xmax * im_width),
                                              int(ymin * im_height), int(ymax * im_height))
                cropped_object = image_np[top:bottom, left:right]
                start_time = time.time()
                input_tensor_classify = tf.convert_to_tensor(np.expand_dims(cropped_object, 0), dtype=tf.uint8)
                classification = classifier_model(input_tensor_classify)
                classification_time = time.time() - start_time
                total_classification_time += classification_time
                classes_classify = classification['detection_classes'][0].numpy().astype(np.int32)
                scores_classify = classification['detection_scores'][0].numpy()
                high_score_index = find_highest_score_index(classes_classify, scores_classify)
                highest_score = scores_classify[high_score_index]
                object_class = classes_classify[high_score_index]
                if highest_score > 0.5 or c_category_index[object_class]['name'] == 'Tank':
                    label = c_category_index[object_class]['name']
                    description = f'{label}: {highest_score:.2f}'
                else:
                    description = 'Unknown'
```

```
                cv2.putText(
                    image_np,
                    description,
                    (left, top + 30),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    1,
                    (0, 255, 0),
                    2,
                    cv2.LINE_AA
                )
                cv2.rectangle(image_np, (left, top), (right, bottom), (255, 0, 0), 2)
            output_image_path = os.path.join(output_folder, f'classified_{image_file}')
            cv2.imwrite(output_image_path,                cv2.cvtColor(image_np,
cv2.COLOR_RGB2BGR))
            num_images += 1
    return   total_detection_time, total_classification_time,num_images
```

Функція crop_objects_from_images

```
def crop_objects_from_images(input_folder, output_folder):
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)
    for filename in os.listdir(input_folder):
        if filename.endswith(".xml"):
            tree = ET.parse(os.path.join(input_folder, filename))
            root = tree.getroot()
            image_filename = root.find("filename").text
            image_path = os.path.join(input_folder, image_filename)
            img = Image.open(image_path)
            img_width, img_height = img.size
```

```
        for idx, obj in enumerate(root.findall("object"), start=1):
            obj_name = obj.find("name").text
            obj_id = 1 if obj_name == "Vehicle" else 2
            xmin = int(obj.find("bndbox/xmin").text)
            xmax = int(obj.find("bndbox/xmax").text)
            ymin = int(obj.find("bndbox/ymin").text)
            ymax = int(obj.find("bndbox/ymax").text)
            cropped_img = img.crop((xmin, ymin, xmax, ymax))
            cropped_filename = f"{obj_name}_{obj_id}_{filename.replace('.xml',
")}_{idx}.jpg"
            cropped_img.save(os.path.join(output_folder, cropped_filename))
            obj_xml = ET.Element("object")
            obj_id_elem = ET.SubElement(obj_xml, "id")
            obj_id_elem.text = str(obj_id)
            obj_name_elem = ET.SubElement(obj_xml, "name")
            obj_name_elem.text = obj_name
            cropped_xml = ET.ElementTree(obj_xml)
            xml_filename = f"{obj_name}_{obj_id}_{filename.replace('.xml',
")}_{idx}.xml"
            cropped_xml.write(os.path.join(output_folder, xml_filename))
    print("COMPLETE")
```

Функція classify_and_compare

```
def classify_and_compare(input_folder, model_dir):
    c_model = tf.saved_model.load(model_dir)
    classifier_model = c_model.signatures['serving_default']
    object_classes = []
    xml_ids = []
    for filename in os.listdir(input_folder):
```

```
        if filename.endswith(".jpg"):
            image_path = os.path.join(input_folder, filename)
            image_np = cv2.imread(image_path)
            image_np = cv2.cvtColor(image_np, cv2.COLOR_BGR2RGB)
            input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0),
dtype=tf.uint8)
            predictions = classifier_model(input_tensor)
            classes_classify =
predictions['detection_classes'][0].numpy().astype(np.int32)
            scores_classify = predictions['detection_scores'][0].numpy()
            high_score_index = find_highest_score_index(classes_classify,
scores_classify, 0.4)
            object_class = classes_classify[high_score_index]
            xml_filename = os.path.splitext(filename)[0] + ".xml"
            xml_path = os.path.join(input_folder, xml_filename)
            if os.path.exists(xml_path):
                tree = ET.parse(xml_path)
                root = tree.getroot()
                xml_id = None
                xml_id = int(root.find("id").text)
                if xml_id is not None:
                    object_classes.append(object_class)
                    xml_ids.append(xml_id)
                else:
                    object_classes.append(-1)
                    xml_ids.append(-1)
    return object_classes, xml_ids
```