МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп'ютерних наук та кібербезпеки

На правах рукопису

КОВАЛЬЧУК ВЛАДИСЛАВ ВАЛЕРІЙОВИЧ

РОЗРОБКА ІНСТРУМЕНТУ ДЛЯ ТРЕНУВАННЯ МОДЕЛЕЙ TENSORFLOW OBJECT DETECTION

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки та інформаційні технології
Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»

Науковий керівник:
ПАСТЕРНАК ЯРОСЛАВ МИХАЙЛОВИЧ,
доктор фізико-математичних наук, професор
кафедри комп'ютерних наук та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол №_____
засідання кафедри комп'ютерних наук
та кібербезпеки
від _____ 2024 р.
Завідувач кафедри

(_____) Гришанович Т. О.

ЛУЦЬК – 2024

ЗМІСТ

ВСТУП

У сучасному світі технології машинного навчання та штучного інтелекту набувають все більшої популярності та застосування у різних сферах життя. Однією з найперспективніших галузей є комп'ютерний зір, який дозволяє автоматично аналізувати та інтерпретувати зображення та відео. Важливим інструментом у цій галузі є моделі для виявлення об'єктів, які використовуються для виявлення та класифікації об'єктів на зображеннях.

Однією з найпотужніших та найпопулярніших платформ для розробки та тренування таких моделей є TensorFlow, створений компанією Google. TensorFlow Object Detection API надає розробникам зручні засоби для побудови, тренування та впровадження моделей виявлення об'єктів. Однак, процес тренування моделей вимагає значних ресурсів та часу, що може ускладнювати його для новачків та невеликих команд розробників.

На сьогоднішній день існує багато різних інструментів та методик для створення моделей виявлення об'єктів, але вони часто є складними у використанні та вимагають значних технічних знань. Більшість з них потребує налаштування великої кількості параметрів, що може стати перешкодою для тих, хто тільки починає свій шлях у цій сфері.

Тому, важливим завданням є створення доступного і простого у використанні інструменту, який би дозволяв швидко та ефективно тренувати моделі для виявлення об'єктів.

Актуальність теми зумовлена тим, що сучасний розвиток інформаційних технологій відкриває нові горизонти для автоматизації та оптимізації процесів у різних сферах життя. Комп'ютерний зір, як одна з найперспективніших галузей штучного інтелекту, не є виключенням.

TensorFlow Object Detection API є потужним інструментом для розробки моделей виявлення об'єктів, але складність його використання може стати бар'єром для новачків. Використання хмарних сервісів, таких як Google Colaboratory, дозволяє швидко налаштувати середовище для тренування моделей

без значних інвестицій у обчислювальні ресурси, роблячи ці технології більш доступними.

Зростання популярності мобільних пристроїв та вбудованих систем вимагає продуктивних та енергоефективних моделей. Формат TensorFlow Lite дозволяє оптимізувати моделі для використання у таких пристроях.

Отже, розробка інструменту для спрощення процесу тренування моделей TensorFlow Object Detection є актуальною задачею, що сприяє зниженню порогу входження для новачків та підвищенню ефективності використання моделей у різних галузях.

Метою дипломної роботи є створення інструменту для тренування кастомних моделей виявлення об’єктів з використанням TensorFlow у середовищі Google Colab.

Для досягнення поставленої мети потрібно виконати такі **завдання**:

– дослідження та аналіз існуючих інструментів для тренування моделей виявлення об’єктів;

– розробка та реалізація інструменту для тренування кастомних моделей з використанням TensorFlow;

– тестування інструменту на реальних даних для оцінки його ефективності та точності;

– реалізація можливості завантаження на пристрій готової натренованої моделі;

Об’єкт дослідження – технології машинного навчання та глибокого навчання, зокрема інструменти для тренування моделей виявлення об’єктів на зображеннях.

Предмет дослідження – процес проєктування, розробки та реалізації інструментів для тренування моделей виявлення об’єктів, включаючи методи та алгоритми машинного навчання, які використовуються для створення таких моделей, а також програмні засоби для їх навчання, тестування та оцінки продуктивності.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ НЕЙРОННИХ МЕРЕЖ ДЛЯ ВИЯВЛЕННЯ ОБ'ЄКТІВ ТА МЕТОДІВ ЇХ НАВЧАННЯ

1.1. Основні поняття про нейронні мережі

Багато завдань, пов'язаних з інтелектом або розпізнаванням шаблонів, надзвичайно складно автоматизувати, але вони, здається, виконуються дуже легко людьми. Наприклад, люди розпізнають різні об'єкти та осмислюють велику кількість візуальної інформації в їхньому оточенні, що, здавалося б, вимагає дуже мало зусиль. Логічно припустити, що обчислювальні системи, які намагаються виконувати подібні завдання, отримають величезну користь від розуміння того, як люди виконують ці завдання, і моделювання цих процесів у межах фізичних обмежень. Це вимагає вивчення та моделювання нейронних мереж.

Людський мозок є зібранням більш ніж 10 мільярдів взаємопов'язаних нейронів. Кожен нейрон є клітиною (Рис. 1.1), яка використовує біохімічні реакції для прийому, обробки та передачі інформації. [8]

Рисунок 1.1 – Схема біологічного нейрона

Деревоподібні мережі нервових волокон, що називаються дендритами, з'єднані з тілом клітини або сомою, де знаходиться ядро клітини. Від тіла клітини відходить одна довга волокниста структура, яка називається аксоном, який врешті-решт розгалужується на нитки і піднитки, і з'єднується з іншими нейронами через синаптичні термінали або синапси. [8]

Передача сигналів від одного нейрона до іншого на синапсах є складним хімічним процесом, в якому специфічні передавальні речовини виділяються з відправного кінця з'єднання. Ефект полягає в підвищенні або зниженні електричного потенціалу всередині тіла приймаючої клітини. Якщо потенціал досягає порогу, по аксону посилається імпульс і клітина "активується".

Штучні нейронні мережі (ШНМ) були розроблені як узагальнення математичних моделей біологічних нервових систем. Перша хвиля інтересу до нейронних мереж (також відомих як конекціоністські моделі або паралельне розподілене оброблення) виникла після введення спрощених нейронів Маккаллоком і Піттсом у 1943 році. [8]

Основні елементи обробки нейронних мереж називаються штучними нейронами (Рис. 1.2), або просто нейронами чи вузлами. У спрощеній математичній моделі нейрона ефекти синапсів представлені вагами зв'язків, які модулюють ефект відповідних вхідних сигналів, а нелінійну характеристику, яку виявляють нейрони, представляє передатна функція. Імпульс нейрона тоді обчислюється як зважена сума вхідних сигналів, перетворених передатною функцією. Здатність до навчання штучного нейрона досягається шляхом налаштування ваг відповідно до обраного алгоритму навчання. [5]

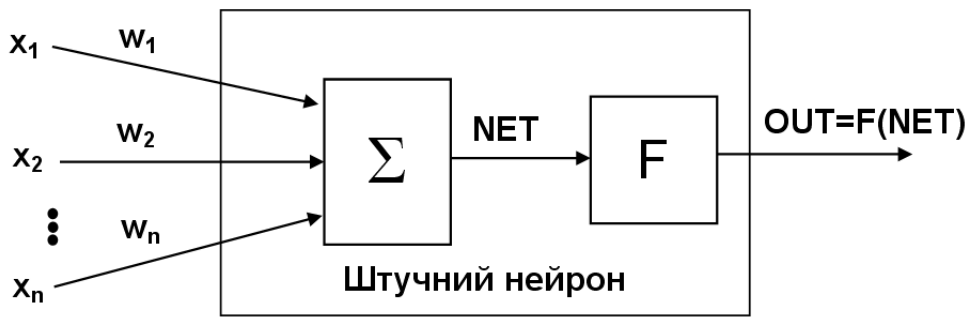

Рисунок 1.2 – Схема штучного нейрона

1.2. Архітектура нейронних мереж

Архітектура нейронних мереж визначає спосіб організації та з’єднання нейронів, що є ключовим аспектом для їх функціонування та ефективності. Базова архітектура нейронної мережі складається з трьох основних типів шарів нейронів [1, 4]:

– вхідний шар нейронної мережі отримує дані з зовнішнього світу. Кожен нейрон цього шару відповідає за один з параметрів вхідного сигналу. Наприклад, у випадку зображення вхідний шар може мати по одному нейрону на кожен піксель;

– приховані шари знаходяться між вхідним і вихідним шарами. Вони виконують основну роботу з обробки даних, передаючи сигнали від одного шару до іншого. Кількість прихованих шарів та кількість нейронів у кожному з них визначають складність і потужність нейронної мережі. Чим більше прихованих шарів, тим складніші патерни може вивчити мережа. У прямопередавальних мережах, таких як багатошаровий перцептрон (MLP), потік сигналів йде від вхідних до вихідних вузлів виключно в прямому напрямку. Обробка даних може здійснюватися через кілька шарів вузлів, але зворотних з’єднань немає, що робить ці мережі простішими для розуміння та навчання;

– вихідний шар відповідає за видачу остаточного результату обробки даних. Кожен нейрон цього шару відповідає за один з параметрів вихідного сигналу. Наприклад, у випадку класифікації зображень вихідний шар може містити стільки нейронів, скільки є можливих класів для класифікації.

На відміну від прямопередавальних мереж, рекурентні нейронні мережі (RNN) містять зворотні зв’язки, що дозволяє їм враховувати попередні стани при обробці нових даних. Це робить RNN надзвичайно ефективними для роботи з послідовними даними, такими як текст або часові ряди, де важливо враховувати контекст або історію попередніх станів. [2]

У рекурентних мережах динамічні властивості мережі відіграють важливу роль. Вони можуть мати різні механізми для управління станами, такі як

довготривала короткочасна пам’ять (LSTM) або ґейтові рекурентні блоки (GRU), що допомагають зберігати та використовувати інформацію протягом тривалих періодів часу. [2]

У деяких випадках значення активації вузлів проходять процес релаксації, в результаті чого мережа еволюціонує до стабільного стану, коли ці активації більше не змінюються. Це може бути корисним у задачах, де необхідно досягти стабільного виходу, який не залежить від подальших змін у вхідних даних.

В інших випадках зміни значень активації вихідних нейронів є значущими, так що динамічна поведінка складає вихідний сигнал мережі. Це особливо важливо у задачах, де вихідний сигнал повинен динамічно змінюватися залежно від поточних вхідних даних, як, наприклад, у системах реального часу.

Архітектура нейронних мереж є основою для їх здатності вивчати та узагальнювати інформацію. Вибір відповідної архітектури залежить від конкретних завдань і типів даних, з якими працює мережа. Прямопередавальні мережі зручні для задач, де важлива проста і зрозуміла передача інформації, тоді як рекурентні мережі краще підходять для обробки послідовних даних з урахуванням їх динаміки.

1.3. Методи навчання нейронних мереж

Нейронні мережі повинні бути налаштовані таким чином, щоб застосування набору вхідних даних призводило до бажаного набору вихідних даних. Існують різні методи налаштування сили зв’язків. Один зі способів полягає в тому, щоб явно задати ваги, використовуючи апріорні знання. Інший спосіб полягає в тому, щоб навчити нейронну мережу, подаючи їй навчальні зразки і дозволяючи їй змінювати свої ваги відповідно до певного правила навчання. [12]

Ситуації навчання в нейронних мережах можна класифікувати на три різні типи:

– навчання з учителем;

- навчання без учителя;
- навчання з підкріпленням.

1.3.1. Навчання з учителем

Навчання з учителем є одним із найпоширеніших методів навчання нейронних мереж. У цьому методі на входи подається вхідний вектор разом із набором бажаних відповідей, по одній для кожного вузла на вихідному шарі. Після проведення прямого проходу обчислюються помилки або розбіжності між бажаною і фактичною відповіддю для кожного вузла на вихідному шарі. Ці помилки потім використовуються для визначення змін ваг у мережі відповідно до діючого правила навчання.

Популярним методом навчання з учителем є алгоритм зворотного поширення помилки, що використовується для нейронних мереж з багатьма шарами. Цей процес полягає в розповсюдженні помилки від вихідного шару до вхідного шару, коригуючи ваги зв’язків відповідно до градієнта функції втрати. Таким чином, модель поступово вдосконалюється, набуваючи здатність точніше прогнозувати вихідні значення на основі вхідних даних. [4]

Важливою характеристикою навчання з учителем є його залежність від наявності якісного набору тренувальних даних з правильними відповідями. Чим більше та точніше ці дані, тим ефективніше може бути навчання моделі. Крім того, важливо правильно обирати архітектуру мережі та параметри навчання, щоб уникнути перенавчання або недонавчання моделі.

Термін "з учителем" походить від того факту, що бажані сигнали на окремих вихідних вузлах надаються зовнішнім учителем.

1.3.2. Навчання без учителя

Навчання без учителя, або самоорганізація в нейронних мережах, відіграє важливу роль у сучасному аналізі даних, де модель вивчає структуру і

закономірності вхідних даних без явного навчання на основі правильних відповідей. Однією з ключових задач цього підходу є кластеризація даних, коли потрібно групувати схожі об’єкти разом на основі їхніх характеристик, незалежно від заздалегідь визначених категорій. [4]

Наприклад, у великих наборах зображень самоорганізація може виявити різні типи об’єктів або сцен, не потребуючи попереднього позначення кожного об’єкту. Модель може автоматично розпізнавати схожість між об’єктами і групувати їх відповідно до їхніх особливостей, що дозволяє швидше та ефективніше аналізувати великі обсяги даних.

Крім того, самоорганізація є важливим інструментом у задачах зниження розмірності даних. Наприклад, у задачах обробки природних мов, модель може автоматично виявляти найбільш важливі слова або теми у текстах, що дозволяє покращити ефективність подальшого аналізу текстової інформації. [12]

Однією з важливих переваг навчання без учителя є можливість розвивати нові методи і підходи до аналізу даних, які не обмежені наявними попередніми знаннями або категоріями. Це дозволяє використовувати нейронні мережі для рішення нових складних завдань, які можуть включати в себе нестандартні або невідомі структури даних.

Такий підхід дозволяє не лише знижувати вартість обробки даних через відсутність необхідності у ручному маркуванні, але й ефективно використовувати потужності сучасних обчислювальних систем для аналізу великих обсягів інформації.

1.3.3. Навчання з підкріпленням

Навчання з підкріпленням є методом навчання нейронних мереж, де модель взаємодіє з оточуючим середовищем з метою максимізації числового сигналу винагороди. У цьому підході навчальний алгоритм не отримує конкретних вказівок, які дії потрібно виконати, як у більшості інших форм машинного навчання. Замість цього, він самостійно визначає оптимальні дії,

експериментуючи з різними стратегіями.

Основними характеристиками навчання з підкріпленням є пошук методом проб і помилок та відкладена винагорода. Це означає, що модель орієнтується на максимізацію загальної винагороди в майбутньому, а не лише на поточній винагороді. Дії, які вона вибирає, можуть впливати на подальші ситуації і майбутні винагороди, що створює складну динаміку в навчанні. [12]

Навчання з підкріпленням є основою для багатьох сучасних систем штучного інтелекту, таких як системи, які використовуються для автономного керування автомобілями, автономних роботів, гри в шахи та інших складних задач, де модель повинна приймати рішення в реальному часі на основі середовища і максимізувати кінцевий результат.

Кожен метод навчання нейронних мереж має свої переваги та області застосування. Навчання з учителем ефективне для задач, де доступні марковані дані. Навчання без учителя корисне для виявлення прихованих структур у великих наборах даних. Навчання з підкріпленням є незамінним для задач, де важлива взаємодія з динамічним середовищем та оптимізація довгострокових винагород. Вибір методу навчання залежить від конкретних потреб і умов задачі, яку необхідно вирішити.

1.4. Типи нейронних мереж

Існують різноманітні типи нейронних мереж, кожен з яких має свої унікальні особливості і застосування в різних областях машинного навчання.

Основні типи нейронних мереж [7, 9]:

- одношарові нейронні мережі;
- багатошарові перцептрони (MLP);
- рекурентні нейронні мережі (RNN);
- згорткові нейронні мережі (CNN);
- рекурентні згорткові нейронні мережі (RCNN);
- автокодувальники (Autoencoder).

1.4.1. Одношарові нейронні мережі

Одношарові нейронні мережі є важливим етапом у розвитку штучних нейронних мереж. Вони відіграють ключову роль у вступі до машинного навчання та нейронних мереж, демонструючи базові принципи функціонування. Такі мережі складаються лише з вхідного шару нейронів, який прямо зв’язаний з вихідним шаром. [7]

Одношарові мережі є досить простими у побудові і розумінні, оскільки вони не використовують прихованих шарів для складнішої обробки інформації. Вони часто використовуються для розв’язання базових задач, які не потребують глибокого аналізу даних або складних взаємозв’язків між вхідними та вихідними даними.

Наприклад, одношарові мережі можуть успішно застосовуватися для класифікації об’єктів на основі простих ознак або для прогнозування значень на основі однієї або декількох вхідних ознак. Вони також можуть використовуватися для вирішення задач регресії, де потрібно здійснити прогноз на основі числових даних. [3]

Однак важливо враховувати обмежену потужність одношарових мереж у порівнянні з більш складними структурами, такими як багатошарові перцептрони чи глибокі згорткові мережі. Вони не здатні ефективно моделювати складні нелінійні залежності через відсутність прихованих шарів, які дозволяють виявляти більш складні патерни та структури в даних.

У практичних застосуваннях одношарові нейронні мережі часто використовуються як базові моделі для порівняння з більш складними алгоритмами. Вони можуть служити важливим етапом для розуміння основ машинного навчання та підготовки до використання більш ефективних і складних нейронних мереж у складних задачах.

1.4.2. Багатошарові перцептрони

Багатошаровий перцептрон (БП) або Multilayer Perceptron (MLP) — це тип штучної нейронної мережі, який складається з щонайменше трьох шарів: вхідного, одного або більше прихованих шарів і вихідного шару. Кожен шар складається з нейронів, які пов’язані між собою з допомогою ваг. [6]

У багатошарових перцептронах інформація передається через кожен шар нейронів, де кожен нейрон зв’язаний з усіма нейронами попереднього і наступного шарів. Це створює можливість для глибокого нелінійного аналізу вхідних даних та виявлення складних патернів, які можуть бути важливі для прогнозування, класифікації або інших завдань обробки інформації. [17]

Одним із прикладів застосування багатошарових перцептронів є розпізнавання образів у комп’ютерному зорі. Наприклад, такі мережі можуть навчатися розпізнавати обличчя на фотографіях. Завдяки складним зв’язкам між нейронами у прихованих шарах, перцептрони здатні до ефективного вивчення унікальних рис та особливостей об’єктів чи явищ у великих наборах даних.

Одним з ключових викликів у роботі з багатошаровими перцептронами є необхідність правильної настройки параметрів мережі та уникнення перенавчання, коли модель втрачає здатність узагальнювати знання на нові дані. Це вимагає ретельного підбору архітектури мережі, функцій активації та методів оптимізації для досягнення оптимальних результатів.

У практичному застосуванні багатошарові перцептрони використовуються для широкого спектру завдань, від обробки природних мов до аналізу фінансових даних. Вони залишаються важливим інструментом у сучасній науці та технологіях, що дозволяє виконувати складні аналізи і здійснювати прогнози на основі великих обсягів даних.

1.4.3. Рекурентні нейронні мережі

Рекурентні нейронні мережі (RNN) є потужним інструментом у сфері машинного навчання. Одна з головних переваг таких мереж полягає у їх здатності ефективно працювати з послідовними даними, такими як тексти, мовлення чи часові ряди. Вони зберігають і використовують інформацію з минулих моментів для прийняття рішень в майбутньому. Це особливо важливо для завдань, що потребують розуміння послідовностей та довгострокових залежностей.

Архітектура RNN включає набір нейронів, які зв’язані між собою зворотніми зв’язками. Кожен нейрон може передавати інформацію наступному і відправляти її назад у часі. Ця особливість дозволяє мережі RNN ефективно керувати контекстуальною інформацією та розв’язувати завдання з урахуванням історії взаємодій між даними. [2]

Одним з найпоширеніших використань RNN є обробка природньої мови. Наприклад, такі мережі можуть бути навчені для автоматичного перекладу текстів, де кожне слово залежить від попередніх у вихідному реченні. Інші застосування включають аналіз текстів, генерацію нових текстів, передбачення наступних слів у послідовності, розпізнавання мови та інші завдання, що вимагають обробки послідовних даних.

Викликами у роботі з RNN є проблеми зникаючого або вибухаючого градієнту, що ускладнює тренування на довгих або складних послідовностях. Для подолання цих проблем використовуються варіанти RNN, такі як Long Short-Term Memory (LSTM) та Gated Recurrent Unit (GRU), які ефективніше управляють інформацією на великих відстанях у часі.

У реальному житті рекурентні нейронні мережі застосовуються в обробці мовленнєвих сигналів для розпізнавання голосу, аналізі часових рядів у фінансових ринках, генерації музики на основі попередніх нот і багатьох інших сценаріях, де важлива робота з послідовними даними та їх взаємозалежностями. [3]

1.4.4. Згорткові нейронні мережі

Згорткові нейронні мережі (CNN) є одним з найпотужніших інструментів у сучасному машинному навчанні, особливо коли йдеться про обробку візуальних даних. CNN використовують спеціалізовану архітектуру, яка дозволяє автоматично виявляти важливі особливості на зображеннях, роблячи їх незамінними для завдань розпізнавання образів, класифікації зображень та обробки відео. Основна перевага згорткових нейронних мереж полягає у здатності зменшувати кількість параметрів у моделі, зберігаючи при цьому здатність до високої точності візуального аналізу. [9]

Архітектура CNN (Рис. 1.3) складається з кількох типів шарів: згорткових, шарів підвибірки (пулінгу) та повнозв'язних шарів. Згорткові шари застосовують фільтри до вхідних даних, створюючи карти ознак, які виділяють різноманітні характеристики зображень, такі як краєчки, текстури та форми. Ці шари виконують роль детекторів ознак, що дозволяє мережі навчатися ієрархії ознак від простих до складних. [18]

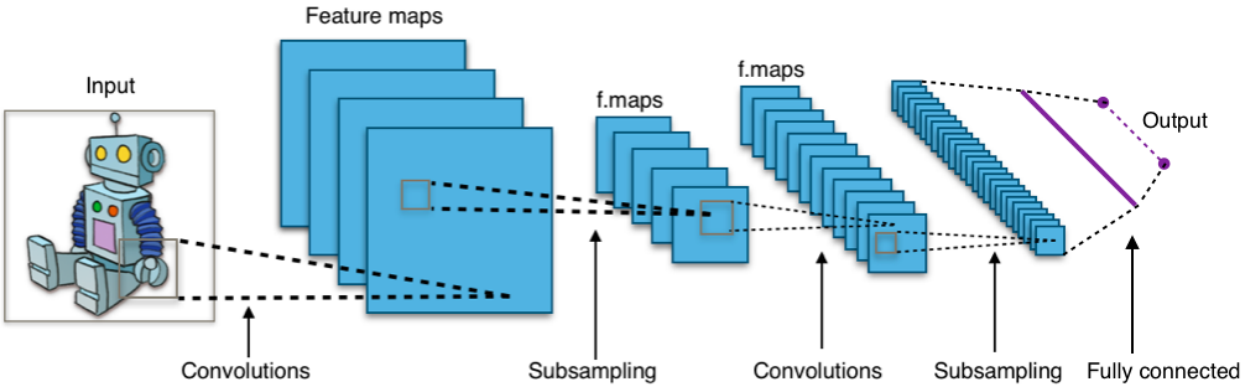

Рисунок 1.3 – Схема архітектури типової згорткової нейронної мережі

Після згорткових шарів ідуть шари підвибірки, які зменшують розмірність даних, зберігаючи найважливіші ознаки. Цей процес зменшує кількість параметрів і обчислювальну складність, а також допомагає уникати перенавчання моделі. Повнозв'язні шари, які зазвичай знаходяться ближче до виходу мережі, виконують функцію класифікатора, обробляючи отримані ознаки

та видаючи остаточний результат. [18]

Одним з прикладів застосування CNN є система розпізнавання облич, яка використовується в безпеці та аутентифікації. Згорткові нейронні мережі також застосовуються в медичній діагностиці для аналізування таких зображень, як рентгенівські знімки, МРТ та КТ, допомагаючи лікарям виявляти патології з високою точністю. Інші важливі застосування включають автопілоти для автомобілів, де CNN використовуються для аналізу зображень з камер, розпізнавання дорожніх знаків, пішоходів і інших об’єктів на дорозі.

Таким чином, згорткові нейронні мережі є основою сучасних систем комп’ютерного зору, що дозволяє автоматизувати та вдосконалювати процеси обробки та аналізу зображень.

1.4.5. Рекурентні згорткові нейронні мережі

Рекурентні згорткові нейронні мережі (RCNN) поєднують у собі переваги згорткових нейронних мереж (CNN) і рекурентних нейронних мереж (RNN), створюючи потужний інструмент для обробки даних, що містять як просторові, так і часові залежності. Цей гібридний підхід дозволяє RCNN ефективно працювати з послідовними візуальними даними, такими як відео, послідовності зображень або інші тимчасові ряди з візуальними характеристиками.

Архітектура RCNN (Рис. 1.4) складається з шарів згортки, які відповідають за виділення просторових ознак з кожного кадру або елементу послідовності, і рекурентних шарів, які аналізують часові залежності між цими ознаками. Згорткові шари виконують функцію детекторів ознак, виділяючи ключові характеристики з кожного окремого кадру, такі як контури, текстури та форми. Ці ознаки потім передаються до рекурентних шарів, що дозволяють моделі враховувати контекстуальну інформацію з попередніх кроків часу, що є критично важливим для розуміння динаміки візуальних даних. [15]

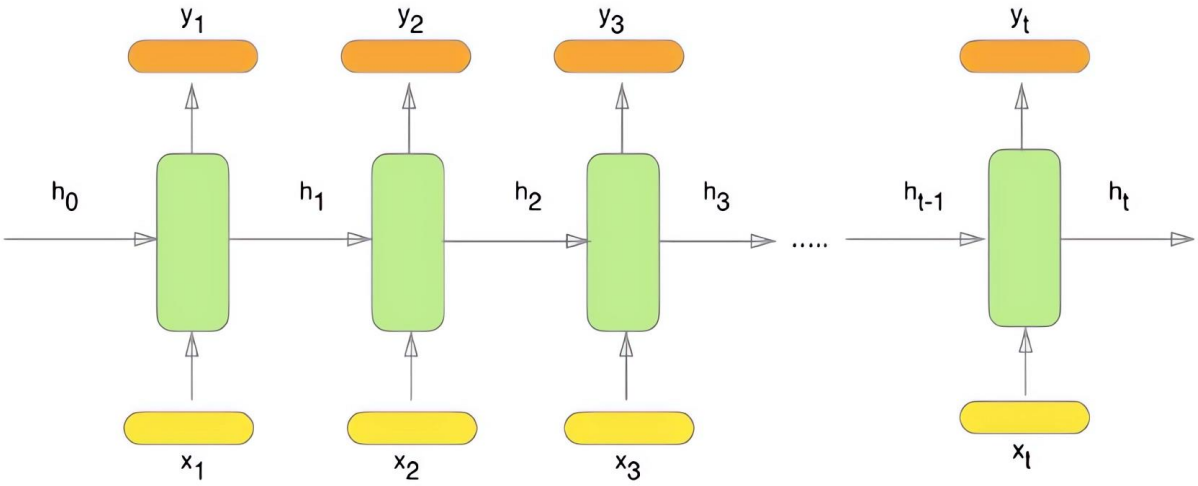

Рисунок 1.4 – Схема архітектури рекурентної згорткової нейронної мережі

Одним із популярних підходів до реалізації RCNN є використання шарів Long Short-Term Memory (LSTM) або Gated Recurrent Unit (GRU), що допомагають ефективніше управляти довгостроковими залежностями та уникати проблем, пов’язаних із зниканням градієнтів під час тренування. Ці шари додають до мережі здатність зберігати та використовувати інформацію з минулих кроків часу, що є важливим для задач, де критичною є послідовність подій. [15]

У сфері медицини RCNN можуть бути використані для аналізу серії медичних зображень, таких як МРТ або ультразвукові дослідження, де важливою є не лише структура окремого зображення, але й зміни, що відбуваються протягом часу. У фінансових ринках RCNN можуть допомагати прогнозувати динаміку цін на основі аналізу як історичних даних, так і поточних ринкових трендів.

Таким чином, рекурентні згорткові нейронні мережі є універсальним інструментом для задач, де важливою є як просторово-часова структура даних, так і контекстуальна інформація. Завдяки поєднанню потужності CNN та RNN, RCNN дозволяють ефективно вирішувати складні задачі обробки послідовних візуальних даних у різних галузях.

1.4.6. Автокодувальники

Автокодувальники представляють собою клас нейронних мереж, які навчаються кодувати вхідні дані в більш стиснену форму і потім відтворювати їх у вихідному форматі. Цей процес називається кодуванням та декодуванням. Автокодувальники складаються з двох основних частин: кодувальника, який стискає вхідні дані, і декодувальника, який відновлює ці дані з стисненої репрезентації. [19]

Кодувальник перетворює вхідні дані на компактне, зменшене представлення, яке називається латентним простором. Метою цього перетворення є виділення найважливіших ознак вхідних даних, що дозволяє зменшити розмірність і усунути зайвий шум. Декодувальник, у свою чергу, використовує це компактне представлення для відновлення вхідних даних. Якщо автокодувальник добре навчається, вихідні дані будуть дуже схожі на оригінальні вхідні дані. [19]

Автокодувальники використовуються в багатьох різних галузях, таких як зменшення розмірності даних, очищення зображень, виявлення аномалій та генерація нових даних. У зменшенні розмірності вони допомагають виділяти найважливіші ознаки з великих наборів даних, що є корисним для візуалізації та аналізу. Наприклад, у комп’ютерному зорі автокодувальники можуть зменшити складність зображень, видаляючи зайвий шум і залишаючи лише найважливіші деталі.

Виявлення аномалій є ще одним важливим застосуванням автокодувальників. Після тренування на нормальних даних, автокодувальник буде погано відтворювати аномальні дані, оскільки ці дані не відповідають звичним патернам. Це дозволяє використовувати автокодувальники для виявлення відхилень у різних сферах, включаючи кібербезпеку, медицину та фінанси.

Генерація нових даних за допомогою автокодувальників також є потужним інструментом. Використовуючи варіаційні автокодувальники (VAE), можна

створювати нові приклади даних, які мають схожі характеристики з вихідним набором. Це відкриває широкі можливості для творчості та інновацій, таких як створення нових зображень, музики або текстів. [17]

Автокодувальники мають просту, але потужну архітектуру (Рис. 1.5), яка дозволяє вирішувати складні задачі в машинному навчанні та штучному інтелекті. Завдяки їх здатності зменшувати розмірність даних і виділяти важливі ознаки, автокодувальники стали незамінним інструментом у багатьох наукових і практичних застосуваннях. [19]

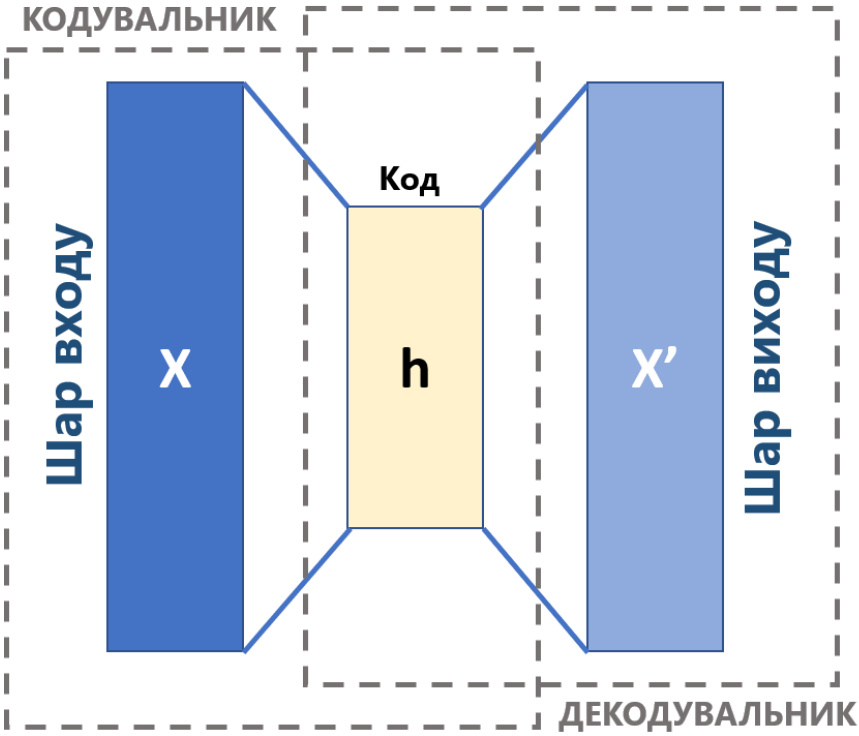

Рисунок 1.5 – Схема архітектури базового автокодувальника

Таким чином, автокодувальники демонструють свою універсальність і потужність у різних задачах, допомагаючи дослідникам і інженерам ефективно обробляти великі обсяги даних і створювати нові, унікальні приклади даних. Завдяки своїй здатності навчатися складним структурам даних і виділяти найважливіші характеристики, автокодувальники продовжують залишатися важливим інструментом у сфері машинного навчання та штучного інтелекту.

1.5. Огляд функціональних можливостей TensorFlow

TensorFlow є однією з провідних бібліотек для машинного навчання та штучного інтелекту, розробленою компанією Google. Ця потужна бібліотека забезпечує інструменти та ресурси для створення та навчання моделей машинного навчання, що можуть виконувати різноманітні завдання, від розпізнавання зображень до обробки природної мови. З моменту свого випуску у 2015 році TensorFlow швидко став популярним серед дослідників та інженерів завдяки своїй гнучкості, масштабованості та здатності працювати на різних платформах.

TensorFlow отримав свою назву від терміну "тензор" (Рис. 1.6), який є узагальненням матриць і векторів. У контексті TensorFlow тензори представляють собою багатовимірні масиви даних, що проходять через графи обчислень. Граф обчислень складається з вузлів, які виконують математичні операції, і ребер, що передають тензори між вузлами. Така архітектура дозволяє ефективно моделювати складні обчислювальні процеси та забезпечує високу продуктивність. [16]

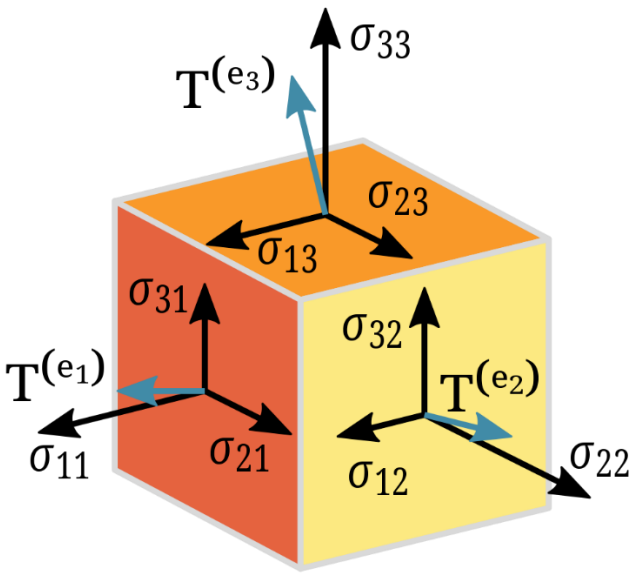

Рисунок 1.6 – Зображення тензора

Однією з ключових особливостей TensorFlow є можливість виконувати обчислення на різних платформах, включаючи процесори (CPU), графічні процесори (GPU) і навіть спеціалізовані апаратні прискорювачі, такі як Tensor Processing Units (TPU). Це забезпечує масштабованість та гнучкість у використанні ресурсів для тренування моделей машинного навчання. [16]

TensorFlow пропонує широкий спектр інструментів для створення та тренування нейронних мереж. Однією з найпопулярніших високорівневих API є Keras, яка забезпечує зручний інтерфейс для швидкої розробки моделей. Keras дозволяє визначати архітектуру мережі, налаштовувати гіперпараметри та тренувати моделі з мінімальними зусиллями.

Для більш досвідчених користувачів TensorFlow надає можливість безпосереднього створення обчислювальних графів з використанням низькорівневих API. Це дозволяє гнучко налаштовувати моделі та оптимізувати їх продуктивність для специфічних задач.

TensorFlow також відомий своєю здатністю працювати з великими обсягами даних. Завдяки підтримці розподілених обчислень, TensorFlow може ефективно обробляти дані, розподілені між декількома машинами. Це особливо важливо для тренування великих моделей на величезних наборах даних, де використання одного комп’ютера може бути неефективним.

TensorFlow пропонує потужні інструменти для зменшення ваги моделі та стиснення даних. Ці можливості є критично важливими для розгортання моделей машинного навчання на пристроях з обмеженими ресурсами, таких як мобільні телефони, вбудовані системи та інтернет речей (IoT).

Однією з ключових технік, які використовуються для зменшення ваги моделі, є прунінг (pruning). Ця методика включає видалення нейронів або зв’язків у нейронній мережі, які мають незначний вплив на вихідні дані. Прунінг дозволяє зменшити кількість параметрів моделі, зберігаючи при цьому її продуктивність. [16]

Процес прунінгу зазвичай включає декілька етапів. Спочатку модель тренується до отримання високої точності. Потім відбувається аналіз важливості

кожного нейрона або зв’язку. Нейрони та зв’язки, що мають низьку важливість, видаляються з моделі. Після цього модель може бути знову тренована для відновлення початкової продуктивності. Цей підхід дозволяє зменшити розмір моделі, що особливо корисно при розгортанні на пристроях з обмеженими обчислювальними ресурсами.

Іншою важливою технікою є квантізація (quantization), яка зменшує точність чисел, що використовуються для зберігання ваг та активацій моделі. Замість використання 32-бітових чисел з плаваючою комою, квантізація дозволяє зменшити ці числа до 16-бітових або навіть 8-бітових. Це не тільки зменшує обсяг пам’яті, необхідний для зберігання моделі, але й прискорює її виконання, оскільки менші числа можуть бути оброблені швидше. [16]

TensorFlow Lite, спеціалізована версія TensorFlow для мобільних пристроїв і вбудованих систем, активно використовує квантізацію для оптимізації моделей. Після квантізації модель може бути виконана на спеціалізованих апаратних прискорювачах, що забезпечує високу швидкість і ефективність роботи.

Стиснення та зменшення ваги моделі мають широкий спектр застосувань. Наприклад, у сфері мобільних додатків ці техніки дозволяють розгортати потужні моделі машинного навчання без значного впливу на продуктивність пристрою. Це може бути використано в додатках для обробки зображень, розпізнавання мови або персоналізованих рекомендацій.

Іншим прикладом є вбудовані системи, де обмежені ресурси потребують оптимізованих моделей для виконання завдань в реальному часі. Наприклад, у сфері автомобільної промисловості моделі машинного навчання можуть використовуватися для аналізу зображень з камер та датчиків для автоматичного керування автомобілем.

Однією з технологій, що підтримується TensorFlow для роботи з великими даними, є Apache Hadoop і Apache Spark. Це дозволяє інтегрувати TensorFlow у розподілені системи обробки даних, що забезпечує високу продуктивність та масштабованість.

Різні формати даних, включаючи зображення, текст, аудіо та відео,

підтримуються цією бібліотекою. Це дозволяє застосовувати TensorFlow у різних галузях, таких як комп'ютерний зір, обробка природної мови та аналіз звукових сигналів. Вбудовані функції для передобробки даних, такі як нормалізація, масштабування та розширення, спрощують підготовку даних для тренування моделей. [16]

Бібліотека забезпечує інструменти для оцінки та валідації моделей, що дозволяє ефективно аналізувати їх продуктивність та знаходити оптимальні параметри. Одним з таких інструментів є TensorBoard, який надає візуалізацію обчислювальних графів, моніторинг метрик тренування та відображення даних у режимі реального часу. Це дає змогу краще розуміти, як працює модель, і виявляти проблеми на ранніх етапах.

Ця технологія використовується у багатьох сферах, таких як медицина, фінанси, автомобільна промисловість, маркетинг та багато інших. Наприклад, у медицині допомагає діагностувати захворювання на основі медичних зображень, аналізувати генетичні дані та розробляти нові методи лікування. У фінансовій сфері вона застосовується для прогнозування ринкових трендів, виявлення шахрайства та автоматизації торгівельних стратегій.

TensorFlow є потужним інструментом для розробки та тренування моделей машинного навчання. Його гнучкість, масштабованість та підтримка різних форматів даних роблять його ідеальним вибором для вирішення широкого спектру задач. Незалежно від того, чи ви працюєте з зображеннями, текстом або аудіо, бібліотека забезпечує необхідні інструменти для успішної реалізації проєктів у сфері штучного інтелекту та машинного навчання. Завдяки своїм можливостям та постійним оновленням, цей інструмент залишається одним з провідних рішень для дослідників та інженерів по всьому світу.

1.6. Огляд та аналіз існуючих інструментів для тренування моделей виявлення об'єктів

На сьогоднішній день існує багато інструментів для тренування моделей

виявлення об’єктів, які різняться за функціональністю, ефективністю та набором можливостей. Аналіз цих інструментів допоможе зрозуміти, які з них найбільше підходять для конкретних завдань і які переваги та недоліки вони мають.

Один з найпопулярніших інструментів для тренування моделей виявлення об’єктів - TensorFlow Object Detection API. Цей інструмент надає зручний інтерфейс для побудови, тренування і валідації моделей, що здатні виявляти об’єкти на зображеннях або відео. Він базується на TensorFlow, що дозволяє використовувати всю потужність цієї бібліотеки для розробки та оптимізації моделей. TensorFlow Object Detection API підтримує різні архітектури, такі як Faster R-CNN, SSD, YOLO, що робить його універсальним інструментом для різноманітних використань.

Ще одним популярним варіантом є YOLO (You Only Look Once), який відомий своєю високою швидкодією та здатністю виявляти об’єкти в реальному часі. Алгоритм YOLO реалізується у вигляді одного інтегрованого проходу через зображення, що робить його особливо ефективним для задач, де важлива швидкість виявлення. Він має кращу роботу з малими об’єктами порівняно з іншими алгоритмами і є популярним серед розробників у сферах відеоспостереження та автомобільної промисловості.

Окрім цього, Darknet і Darkflow від компанії OpenCV є ще одними інструментами, які використовуються для тренування моделей виявлення об’єктів, зокрема за допомогою алгоритму YOLO. Вони надають можливість розробляти і адаптувати моделі для різних завдань, враховуючи специфіку даних і обчислювальні потреби.

За кожним інструментом для тренування моделей виявлення об’єктів стоять певні переваги та обмеження. Вибір підходящого інструменту залежить від конкретного завдання, характеристик даних, обчислювальних ресурсів і вимог до продуктивності. Розглядати ці інструменти з точки зору їх особливостей і можливостей дозволяє ефективно використовувати їх потенціал у практичних задачах з виявлення об’єктів.

1.6.1. Google Cloud AutoML

Одним з інноваційних інструментів для тренування моделей виявлення об’єктів є Google Cloud AutoML, який надає можливість розробникам і компаніям створювати та вдосконалювати моделі штучного інтелекту без необхідності в глибоких знаннях у галузі машинного навчання. Google Cloud AutoML базується на потужних обчислювальних ресурсах та алгоритмах компанії Google.

Цей інструмент відкриває можливості для розробників з різних галузей, у тому числі для тих, хто не має глибоких знань у сфері машинного навчання. Він пропонує інтуїтивно зрозумілий інтерфейс, який дозволяє завантажувати дані, маркувати їх, вибирати моделі та тренувати їх за допомогою кількох клацань миші. Такий підхід робить Google Cloud AutoML доступним для широкого кола користувачів, від малих стартапів до великих корпорацій. [10]

Однією з ключових переваг Google Cloud AutoML є його висока швидкість тренування моделей та автоматизація багатьох етапів розробки. Він використовує передові технології автоматичного вибору гіперпараметрів, оптимізації архітектур моделей та зменшення необхідного для тренування часу. Це особливо важливо для проєктів, де швидкість розгортання та ефективність моделей мають критичне значення.

Крім того, Google Cloud AutoML підтримує різні формати даних, включаючи зображення, текст та аудіо, що розширює його застосування на різних ринках і галузях. [10]

Незважаючи на свої переваги, Google Cloud AutoML також має деякі обмеження та виклики. Один з таких викликів полягає у вартості використання. Такі передові технології можуть вимагати значних витрат на обчислювальні ресурси та підтримку, що може зробити їх недоступними для малих підприємств або стартапів з обмеженим бюджетом. Крім того, існує питання конфіденційності даних, особливо коли йдеться про використання облікових записів користувачів для тренування моделей.

У підсумку, Google Cloud AutoML (Рис. 1.7) є потужним інструментом для розробки та тренування моделей виявлення об’єктів, який відкриває нові можливості для бізнесу та науки. Його здатність спрощувати процеси тренування моделей, швидкість розгортання та підтримка різних форматів даних роблять його важливим інструментом у сучасному світі штучного інтелекту. Проте необхідно ураховувати його вартість при виборі для конкретного проєкту чи дослідження.

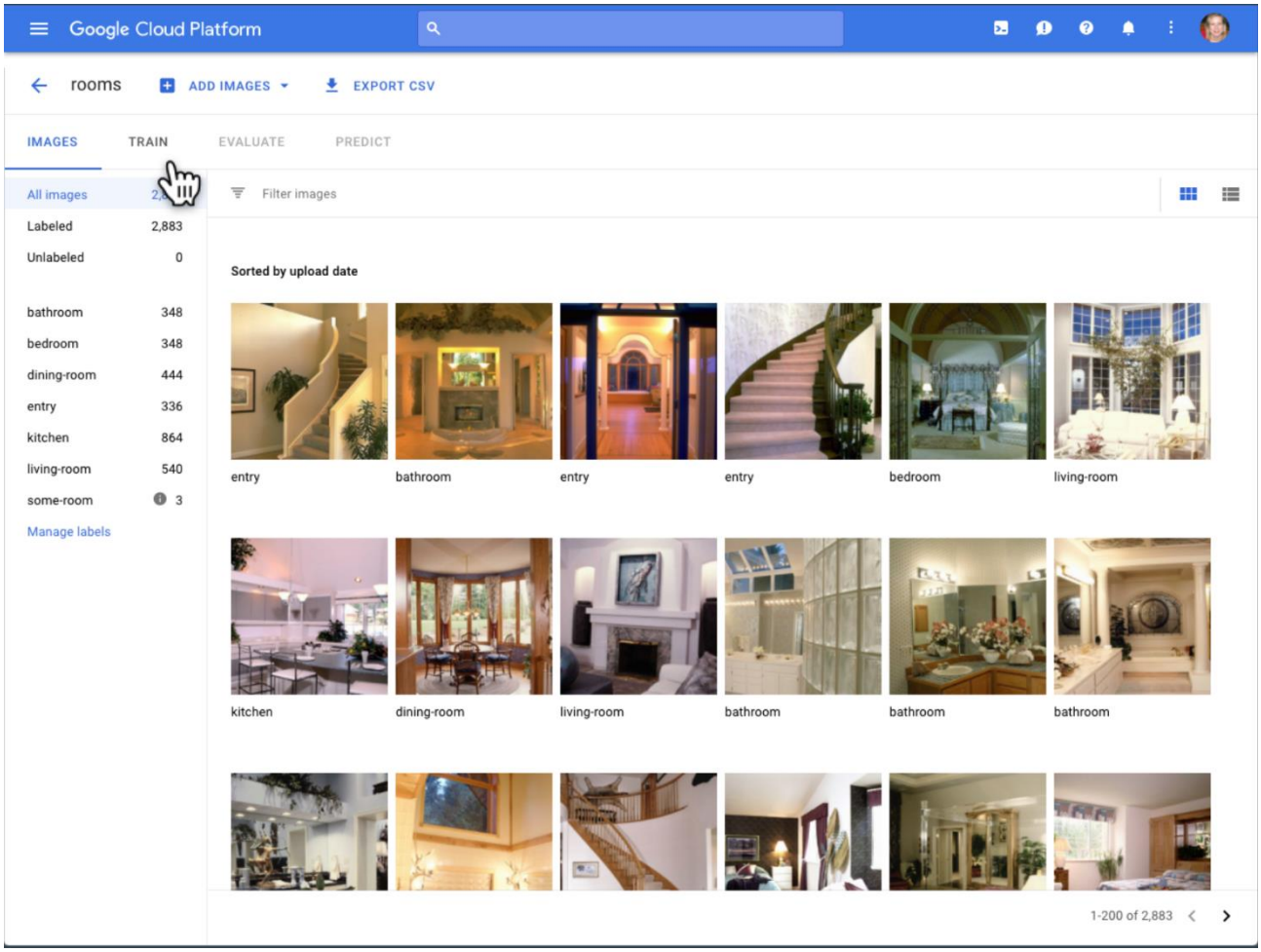

Рисунок 1.7 – Знімок екрана головного меню Google Cloud AutoML

1.6.2. Amazon Rekognition

Amazon Rekognition - це один з передових інструментів для виявлення об’єктів та розпізнавання зображень, який надається Amazon Web Services (AWS). Цей сервіс пропонує широкий спектр можливостей для роботи з

візуальним контентом, що включає аналіз зображень та відео, виявлення об’єктів і сцен, розпізнавання тексту на зображеннях, а також обробку потокових відео.

Цей сервіс базується на нейронних мережах і використовується для автоматичного аналізу великих обсягів візуальної інформації з метою виділення ключових об’єктів та патернів. Однією з ключових можливостей Amazon Rekognition є здатність розпізнавати обличчя на зображеннях і відео, що дозволяє використовувати його для реалізації систем ідентифікації або аналізу емоцій. Наприклад, цей інструмент застосовується для впровадження систем контролю доступу, які відкривають двері або надають доступ на основі розпізнавання облич людей. Крім того, Amazon Rekognition дозволяє виявляти та класифікувати об’єкти на зображеннях і відео. Ця функціональність корисна у таких галузях як ритейл (для виявлення товарів на полицях магазинів), медицина (для аналізу медичних зображень) та автомобільна промисловість (для автоматичного виявлення дорожніх знаків і інших об’єктів на дорогах).

Процес роботи з Amazon Rekognition включає кілька ключових етапів. Починаючи з завантаження зображень або відео до AWS, користувачі можуть використовувати API для виклику потрібних функцій, таких як розпізнавання облич, виявлення об’єктів або аналіз тексту. Результати аналізу можуть бути представлені у вигляді JSON-об’єктів, що містять інформацію про знайдені об’єкти, їх координати на зображенні, а також інші метадані, що дозволяють подальший аналіз і обробку результатів. [11]

Одним із важливих аспектів використання Amazon Rekognition є його готовність до інтеграції з іншими сервісами AWS, такими як Amazon S3 для зберігання великих обсягів візуальних даних або Amazon Lambda для автоматичного запуску функцій обробки візуального контенту. Це дозволяє створювати повністю автоматизовані рішення для обробки зображень та відео на масштабованих інфраструктурах AWS. [11]

Візуалізація цього процесу може бути зображена через схему, яка показує взаємодію між AWS, користувачем і вхідними даними, які проходять через процес аналізу з використанням Amazon Rekognition (Рис. 1.8).

Таким чином, Amazon Rekognition є потужним інструментом для виявлення об’єктів і аналізу зображень, що знаходить своє застосування у багатьох сферах, від безпеки і медіааналітики до автоматизації бізнес-процесів. Його широкі можливості та інтеграція з іншими сервісами AWS роблять його популярним вибором серед розробників і компаній, що шукають надійне рішення для роботи з візуальним контентом у хмарному середовищі.

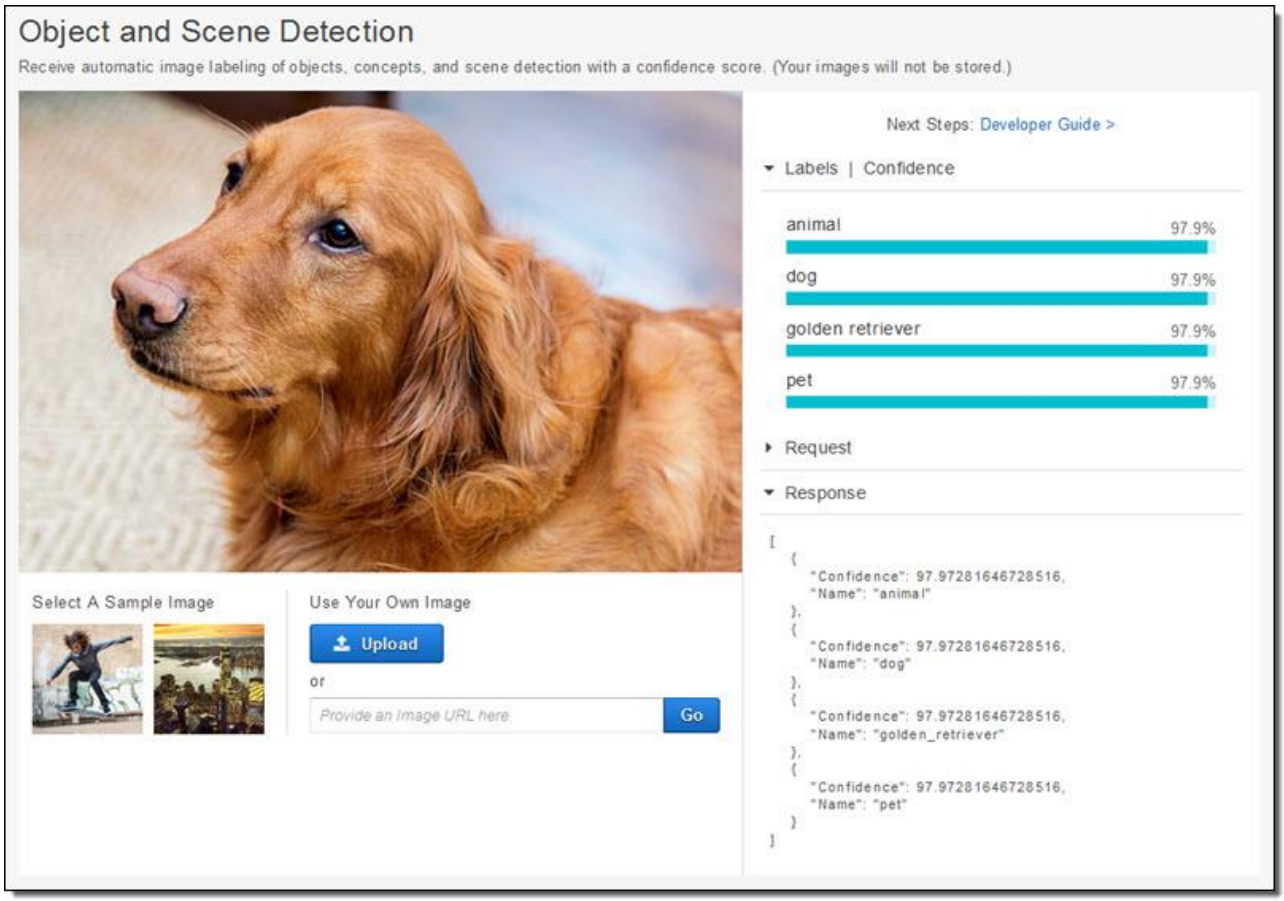

Рисунок 1.8 – Знімок виявлення об’єкта та сцени в Amazon Rekognition

РОЗДІЛ 2
РОЗРОБКА ТА РЕАЛІЗАЦІЯ ІНСТРУМЕНТУ ДЛЯ ТРЕНУВАННЯ КАСТОМНИХ МОДЕЛЕЙ ВИЯВЛЕННЯ ОБ'ЄКТІВ

2.1. Постановка задачі, призначення та вимоги до розробки

Зі швидким розвитком технологій штучного інтелекту та комп'ютерного зору, з'являється потреба у ефективних інструментах для виявлення об'єктів на зображеннях. TensorFlow Object Detection API є одним з провідних рішень у цій галузі, що дозволяє створювати, тренувати та впроваджувати моделі для об'єктного виявлення. Однак, процес тренування таких моделей може бути досить складним та вимагати значних зусиль, особливо для початківців.

Основною задачею даної бакалаврської роботи є розробка інструменту для тренування моделей TensorFlow Object Detection (Рис. 2.1), який буде зручним у використанні та забезпечить можливість автоматизації багатьох рутинних процесів. Цей інструмент має полегшити підготовку даних, конфігурацію моделей, процес тренування та оцінювання результатів.

Розроблений інструмент призначений для ентузіастів у галузі машинного навчання та комп'ютерного зору, які прагнуть створити власні моделі для виявлення об'єктів. Використання цього інструменту дозволить значно зекономити час та зусилля, необхідні для підготовки даних та налаштування моделей.

Інструмент забезпечує налаштування параметрів тренування, та автоматизує процес конвертації моделі у формат TensorFlow Lite. Окрім того, передбачено можливість оцінювання якості тренованої моделі за допомогою обчислення середньої точності (mAP).

Вимоги до розробки:

– система повинна забезпечувати користувачам можливість завантаження власних наборів даних для тренування моделей виявлення об'єктів у середовищі Google Colab;

– повинна бути передбачена функціональність для налаштування параметрів тренування моделей, включаючи вибір архітектури моделі, параметри навчання та критерії зупинки;

– користувачі повинні мати можливість переглядати та аналізувати результати тренування моделей, включаючи візуалізацію точності, втрат та інших метрик;

– система повинна надавати інструменти для тестування натренованих моделей на нових наборах даних з можливістю візуалізації результатів виявлення об’єктів.

– повинен бути забезпечений зручний інтерфейс для збереження та завантаження моделей, а також для експорту моделей у різні формати для подальшого використання.

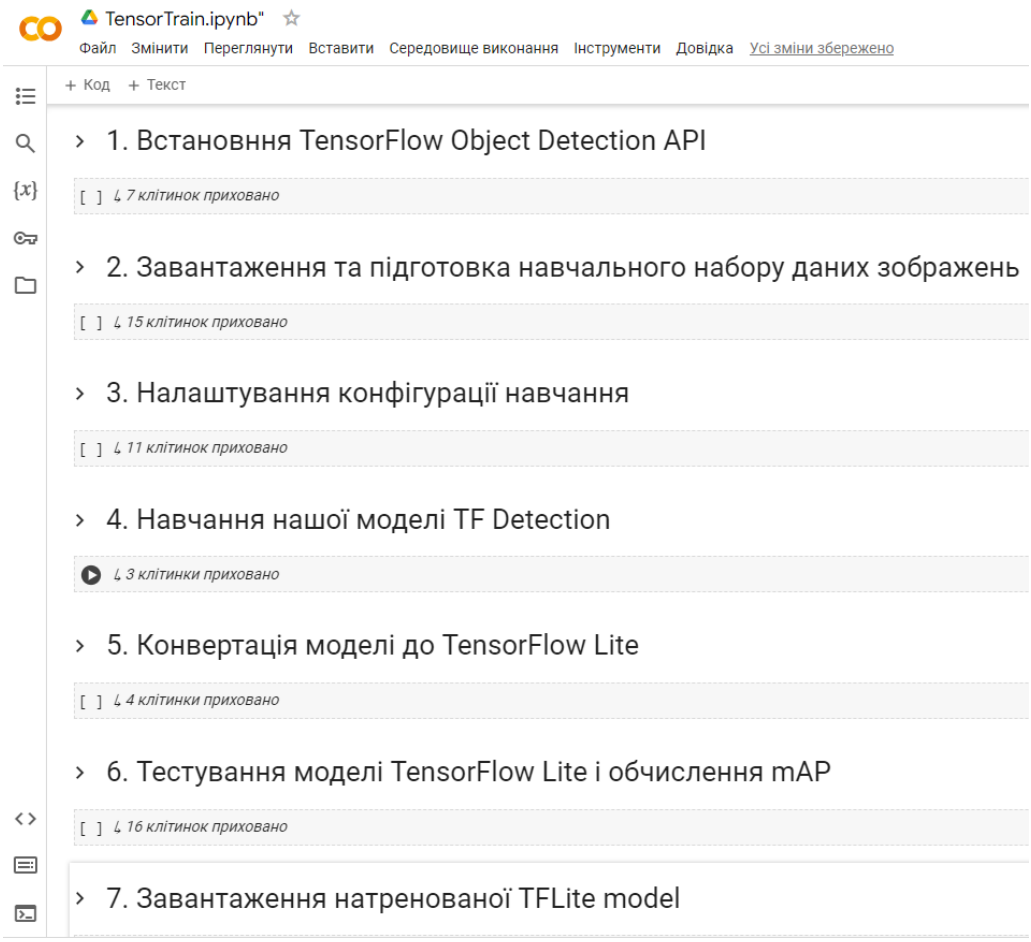

Рисунок 2.1 – Загальна структура проєкту в середвищі Google Colab

Таким чином, розробка інструменту для тренування моделей TensorFlow Object Detection допоможе значно спростити та автоматизувати процес створення ефективних моделей для виявлення об’єктів.

2.2. Загальна структура проєкту

Основною метою цього проєкту є спрощення та автоматизація процесу тренування моделей, що дозволить зосередитися на удосконаленні моделей та аналізі їх результатів, а не витрачати значний час на рутинні завдання.

Проєкт знаходиться на Google Colab та складається з кількох основних етапів, кожен з яких вирішує конкретні задачі, пов’язані з тренуванням моделей.

На початковому етапі користувач завантажує навчальні зображення. Спочатку на локальному комп’ютері потрібно заархівувати всі свої навчальні зображення та XML-файли з анотаціями до них в одну папку під назвою «images.zip». Файли мають бути безпосередньо в папці zip або у вкладеній папці, як показано (Рис 2.2). Цей процес є критично важливим, оскільки від якості даних залежить точність майбутньої моделі.

```
images.zip
-- images
  -- img1.jpg
  -- img1.xml
  -- img2.jpg
  -- img2.xml
  ...
```

Рисунок 2.2 – Приклад вмісту набору даних images.zip

Після підготовки даних, інструмент автоматично завантажує необхідні залежності для TensorFlow Object Detection. Це забезпечує правильну конфігурацію середовища та усуває можливі помилки, пов’язані з несумісністю версій програмного забезпечення.

Далі відбувається підготовка набору даних зображень для тренування. Інструмент виконує попередню обробку зображень, що включає розділення їх на папки train, validation, та test та інші необхідні операції. Це дозволяє підготувати дані у форматі, який оптимально підходить для тренування моделі.

Одним з ключових етапів є налаштування конфігурації тренування. Інструмент надає користувачам можливість визначати різні параметри тренування, такі як кількість кроків, розмір вибірки та вибір попередньо навченої моделі TensorFlow для завантаження з TensorFlow 2 Object Detection Model Zoo. Правильний вибір цих параметрів впливає на ефективність навчання та якість кінцевої моделі.

Після налаштування параметрів розпочинається процес тренування моделі. Інструмент використовує навчальні дані та обрані параметри для оптимізації вагів моделі. Під час тренування користувачі можуть відстежувати прогрес за допомогою графіків, що показують зміну точності та втрат на різних етапах навчання. Це дозволяє своєчасно виявляти проблеми та вносити необхідні корективи.

Після завершення тренування модель конвертується у формат TensorFlow Lite, що дозволяє її використовувати на мобільних та вбудованих пристроях. Цей етап є важливим для забезпечення широкої застосовності моделі в реальних умовах, де обчислювальні ресурси можуть бути обмеженими.

Останнім етапом є оцінювання якості тренованої моделі. Інструмент надає можливість обчислення показників середньої точності (mAP) та візуалізує результати. Це допомагає оцінити ефективність моделі та визначити області, які потребують покращення.

2.3. Вибір моделі розробки

Для успішної реалізації проєкту з тренування моделей TensorFlow Object Detection, важливо вибрати модель розробки, яка дозволить ефективно організувати процеси, забезпечить гнучкість і простоту в управлінні.

Враховуючи специфіку завдання, доцільно зупинитися на каскадній моделі розробки, відомій також як «водоспад» (Рис. 2.3).

Рисунок 2.3 – Каскадна або водоспадна модель

Каскадна модель передбачає послідовне виконання етапів розробки, де кожен етап починається після завершення попереднього. Такий підхід дозволяє чітко структурувати процес розробки, уникнути плутанини та забезпечити детальне планування на кожному етапі. У випадку з нашим проєктом, це забезпечить чіткість у виконанні задач та полегшить управління проєктом.

Першим етапом є аналіз вимог. На цьому етапі визначаються цілі та завдання проєкту, вимоги до функціоналу та характеристик системи. Для нашого інструменту тренування моделей важливо чітко розуміти, які функції мають бути реалізовані, які дані будуть використовуватися та які результати очікуються. Це дозволить уникнути помилок у майбутньому та забезпечити відповідність кінцевого продукту вимогам користувачів.

Другий етап - проєктування системи. На цьому етапі розробляється архітектура інструменту, визначаються основні компоненти та їх взаємодія. Важливо передбачити модульну структуру, де кожен модуль відповідає за

конкретну функцію, наприклад, обробку даних, тренування моделі, оцінювання якості та конвертацію у формат TensorFlow Lite. Така архітектура полегшить подальшу розробку та тестування системи.

Наступний етап - реалізація. На цьому етапі відбувається безпосередня розробка інструменту. Кожен модуль розробляється окремо, тестується та інтегрується у загальну систему. Використання каскадної моделі дозволяє чітко планувати порядок розробки модулів та уникнути непорозумінь між різними етапами розробки.

Четвертий етап - тестування. Після завершення реалізації кожного модуля та їх інтеграції у загальну систему відбувається тестування. Важливо перевірити коректність роботи кожного модуля окремо, а також системи в цілому. Це дозволить виявити та виправити помилки перед введенням інструменту в експлуатацію.

Завершальний етап - впровадження та супровід. Після успішного тестування інструмент вводиться в експлуатацію. Важливо забезпечити належну підтримку користувачів, швидко реагувати на їх запити та вносити необхідні зміни для покращення роботи інструменту. Постійний моніторинг та оновлення системи дозволять підтримувати її актуальність та ефективність.

Використання каскадної моделі розробки для проєкту тренування моделей TensorFlow Object Detection дозволяє чітко структурувати процес розробки, забезпечити послідовність виконання завдань та високу якість кінцевого продукту. Такий підхід є ефективним для проєктів, де важливо ретельно планувати кожен етап та уникати повторних робіт.

2.4. Обґрунтування вибору інструментальних засобів розробки

При виборі інструментальних засобів для розробки інструменту тренування моделей TensorFlow Object Detection, важливо було врахувати кілька ключових факторів: зручність використання, функціональність, сумісність з існуючими технологіями та можливості для масштабування. З урахуванням цих

аспектів, ми зупинили свій вибір на мові програмування Python, середовищі розробки Google Colab, бібліотеці TensorFlow.

Python був обраний як основна мова програмування через його популярність у сфері машинного навчання та глибокого навчання. Ця мова відрізняється простотою та читабельністю синтаксису, що значно полегшує розробку та підтримку коду. Крім того, Python має широкий спектр бібліотек та інструментів для машинного навчання, що робить його ідеальним вибором для створення складних моделей та роботи з великими обсягами даних. Бібліотека TensorFlow, яка також написана на Python, дозволяє ефективно тренувати та застосовувати нейронні мережі для задач виявлення об’єктів.

Вибір середовища розробки Google Colab був обґрунтований кількома факторами. По-перше, це безкоштовний інструмент, який надає доступ до потужних обчислювальних ресурсів, включаючи графічні процесори (GPU) та тензорні процесори (TPU). Це дозволяє значно прискорити процес тренування моделей, що особливо важливо при роботі з великими наборами даних. По-друге, Google Colab підтримує інтеграцію з Google Drive, що полегшує зберігання та доступ до даних.

TensorFlow був вибраний як основна бібліотека для машинного навчання завдяки його потужним можливостям та широкій підтримці спільнотою розробників. TensorFlow надає багатий набір інструментів для створення, тренування та оптимізації моделей нейронних мереж. Він підтримує різні типи нейронних мереж, включаючи згорткові нейронні мережі (CNN), які є основою для задач комп’ютерного зору. Крім того, TensorFlow має можливість експорту моделей у формат TensorFlow Lite, що дозволяє використовувати їх на мобільних та вбудованих пристроях.

2.4.1. TensorFlow

TensorFlow є однією з найпотужніших та найпопулярніших бібліотек для машинного навчання і глибокого навчання. Вона була розроблена компанією

Google і вперше представлена публіці у 2015 році. Завдяки своїм широким можливостям та відкритому коду, TensorFlow швидко здобула визнання у науковій спільноті та серед інженерів, що працюють у галузі штучного інтелекту. [16]

Основною особливістю TensorFlow є її здатність працювати з багатовимірними масивами даних (тензорами) та виконувати складні математичні операції на них. Назва бібліотеки походить саме від слів "tensor" (тензор) та "flow" (потік), що відображає основну концепцію роботи з потоками даних. TensorFlow дозволяє будувати та тренувати різноманітні моделі нейронних мереж, включаючи згорткові нейронні мережі (CNN), рекурентні нейронні мережі (RNN) та інші архітектури, що використовуються для задач комп’ютерного зору, обробки природної мови та інших напрямків.

Одна з ключових переваг TensorFlow – це його гнучкість. Бібліотека підтримує різні рівні абстракції, що дозволяє використовувати її як новачкам, так і досвідченим розробникам. Наприклад, високорівневий API Keras, який є частиною TensorFlow, надає простий та інтуїтивно зрозумілий інтерфейс для швидкого створення та тренування моделей. У той же час, більш низькорівневий API TensorFlow дає змогу здійснювати тонке налаштування моделі та виконувати оптимізацію на рівні окремих операцій.

TensorFlow також забезпечує високий рівень продуктивності. Завдяки підтримці апаратного прискорення, бібліотека може ефективно використовувати ресурси графічних процесорів (GPU) та тензорних процесорів (TPU). Це дозволяє значно прискорити процес тренування моделей, що особливо важливо при роботі з великими наборами даних та складними архітектурами нейронних мереж. Крім того, TensorFlow підтримує розподілене обчислення, що дає змогу масштабувати процес тренування на декілька машин, що працюють одночасно. [16]

Ще однією важливою особливістю TensorFlow є її екосистема інструментів та бібліотек. Наприклад, TensorFlow Lite дозволяє оптимізувати та розгортати моделі на мобільних та вбудованих пристроях, забезпечуючи високу

продуктивність та малий розмір моделі. TensorFlow Serving забезпечує ефективне розгортання моделей на сервері для обробки запитів у реальному часі. TensorFlow.js дозволяє використовувати моделі безпосередньо у веб-браузері, що відкриває нові можливості для створення інтерактивних веб-додатків.

2.4.2. Мова програмування Python

Python є однією з найпопулярніших мов програмування у світі, і це не випадково. Її популярність зумовлена простотою синтаксису, багатим набором бібліотек та універсальністю. Python став ключовим інструментом для розробників у багатьох галузях, включаючи науку про дані, машинне навчання, веб-розробку, автоматизацію та багато інших. У контексті розробки інструментів для тренування моделей TensorFlow Object Detection, Python виступає ідеальним вибором завдяки своїм численним перевагам.

Однією з головних причин вибору Python є його простота та читабельність. Синтаксис Python нагадує природну мову, що робить його легким для розуміння та написання коду навіть для початківців. Це дозволяє розробникам швидко створювати прототипи та експериментувати з різними підходами до вирішення задач. Для складних проєктів, таких як розробка інструментів для машинного навчання, ця властивість Python значно скорочує час розробки та дозволяє зосередитися на алгоритмах та логіці, а не на деталях реалізації. [14]

Ще однією важливою перевагою Python є його багата екосистема бібліотек та фреймворків. Для машинного навчання та глибокого навчання існує безліч спеціалізованих бібліотек, таких як TensorFlow, Keras, PyTorch, Scikit-learn та інші. Ці бібліотеки забезпечують готові до використання інструменти та алгоритми, що дозволяє розробникам зосередитися на специфіці їхньої задачі. У випадку нашого проєкту, використання TensorFlow у поєднанні з Python дає можливість ефективно будувати, тренувати та розгортати моделі для виявлення об’єктів.

Python також підтримує інтерактивне програмування, що особливо

корисно для досліджень та експериментів. Інструменти, такі як Jupyter Notebooks, дозволяють писати код, виконувати його та бачити результати у режимі реального часу. Це створює зручне середовище для аналізу даних, візуалізації результатів та відладки моделей. У контексті нашого проєкту, використання Google Colab, який підтримує Jupyter Notebooks, забезпечує потужне середовище для розробки та тестування моделей.

Гнучкість Python дозволяє легко інтегрувати різні компоненти системи та працювати з різними форматами даних. Python підтримує різноманітні бібліотеки для роботи з зображеннями, такими як OpenCV та Pillow, що дозволяє легко обробляти та підготувати дані для тренування моделей. Крім того, Python має потужні інструменти для аналізу та візуалізації даних, такі як Pandas, NumPy та Matplotlib, що допомагає зрозуміти характеристики даних та оцінити результати моделі. [14]

2.4.3. Середовище розробки Google Colab

Google Colaboratory, або просто Colab, є хмарним сервісом, створеним Google Research, що дозволяє користувачам писати та виконувати вихідний код безпосередньо у браузері. Це інтегроване середовище розробки (IDE), засноване на Jupyter Notebook, яке спеціалізується на задачах машинного навчання, аналізу даних та освітніх проєктах. Colab підтримує мову програмування Python і надає обчислювальні ресурси для редагування та тестування коду, включаючи можливість використання графічних процесорів (GPU) та тензорних процесорів (TPU). Це середовище є безкоштовним і не вимагає встановлення Jupyter, пропонуючи розробникам зручність роботи у віртуальному середовищі. [13]

Однією з головних переваг Google Colab є його доступність та простота використання. Оскільки це веб-додаток, користувачі можуть почати працювати з ним без необхідності встановлення додаткового програмного забезпечення на свої комп’ютери. Все, що потрібно, — це обліковий запис Google. Інтерфейс Colab схожий на Jupyter Notebook, що дозволяє писати та виконувати код, а

також документувати процес розробки у форматі інтерактивних блокнотів. Це особливо корисно для навчання та спільної роботи над проєктами.

Підтримка великої кількості бібліотек та фреймворків для машинного навчання є ще однією важливою перевагою Google Colab. Завдяки вбудованій підтримці TensorFlow, PyTorch, Keras та інших популярних інструментів, розробники можуть легко експериментувати з різними технологіями та інтегрувати їх у свої проєкти. Для нашого проєкту, де використовується TensorFlow, ця підтримка є надзвичайно цінною, оскільки вона дозволяє ефективно будувати, тренувати та розгортати моделі для виявлення об’єктів.

Використання Google Colab значно прискорює процес тренування моделей завдяки доступу до потужних обчислювальних ресурсів. Можливість використовувати GPU та TPU дозволяє суттєво зменшити час, необхідний для тренування складних нейронних мереж. Це особливо актуально при роботі з великими наборами даних, де звичайний центральний процесор (CPU) може бути недостатньо швидким. Крім того, Google Colab автоматично управляє обчислювальними ресурсами, забезпечуючи оптимальну продуктивність без потреби в ручному налаштуванні. [13]

Google Colab також забезпечує зручний доступ до збереження та обміну даними завдяки інтеграції з іншими сервісами Google, такими як Google Drive. Це дозволяє легко зберігати, завантажувати та ділитися даними та результатами експериментів. Наприклад, набори даних можуть бути збережені на Google Drive та легко імпортовані до середовища Colab для подальшої обробки та аналізу. Це значно спрощує управління даними та забезпечує зручний доступ до них з будь-якого місця.

Коли користувач отримує доступ до Colab через обліковий запис Gmail, він отримує віртуальну машину, ізольовану від інших користувачів та ресурсів. Це дозволяє відновлювати віртуальну машину до початкового стану у випадку проблем. Варто зазначити, що хоча машини видаляються після певного періоду бездіяльності, блокноти можна зберегти в Google Drive або завантажити локально у форматі Jupyter (.ipynb). Це забезпечує надійне збереження

результатів роботи та можливість продовжувати роботу в будь-який момент.

Google Colab також забезпечує високу надійність та безпеку даних. Усі обчислення виконуються на серверах Google, що гарантує захист даних та високу доступність сервісу. Крім того, автоматичне збереження змін у блокнотах запобігає втраті даних у разі непередбачених обставин. [13]

2.5. Особливості програмної реалізації

Процес розробки та реалізації програмного забезпечення для тренування моделей TensorFlow Object Detection має ряд особливостей. Основною платформою для виконання цього проєкту було обрано Google Colab, що надає потужні обчислювальні ресурси та інтеграцію з інструментами Google.

Першим кроком є встановлення необхідних бібліотек та інструментів. У випадку з TensorFlow Object Detection API, як показано на прикладі у зображенні (Рис. 2.4), процес починається з клонування відповідного репозиторію з GitHub і встановлення додаткових залежностей, таких як CUDA для використання обчислювальних можливостей графічного процесора.

Для ефективного навчання моделей машинного навчання критично важливо правильно розділити дані на три основні набори: train, validation, та test. Це розділення забезпечує можливість не лише навчити модель, а й перевірити її здатність узагальнювати знання на нових даних, що є ключовим для створення надійних та точних систем.

Набір зображень, які використовуються для навчання моделі, називається train. На кожному кроці навчання нейронна мережа отримує партію зображень з цього набору. Використовуючи ці дані, мережа робить передбачення про класи та розташування об’єктів на зображеннях. Алгоритм навчання обчислює втрати, що представляють собою різницю між передбаченнями моделі та фактичними мітками, і коригує ваги мережі за допомогою методу зворотного поширення помилки. Цей процес повторюється багаторазово, поки модель не досягне прийнятного рівня точності на тренувальних даних.

```
# Клонування репозиторію моделей tensorflow з GitHub
!pip uninstall Cython -y
!git clone --depth 1 https://github.com/tensorflow/models

# Копіювання файлів налаштувань у папку models/research
%%bash
cd models/research/
protoc object_detection/protos/*.proto --python_out=.

# Зміна файлу setup.py для встановлення репозиторію tf-models-official,
# орієнтованого на TF версії 2.8.0
import re
with open('/content/models/research/object_detection/packages/tf2/setup.py') as f:
    s = f.read()

with open('/content/models/research/setup.py', 'w') as f:
    # Установка шляху до fine_tune_checkpoint
    s = re.sub('tf-models-official>=2.5.1',
               'tf-models-official==2.8.0', s)
    f.write(s)

# Встановлення Object Detection API
!pip install pyyaml==5.3
!pip install /content/models/research/
!pip install tensorflow==2.8.0

# Встановлення CUDA версії 11.0
!pip install tensorflow_io==0.23.1
!wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/cuda-
!mv cuda-ubuntu1804.pin /etc/apt/preferences.d/cuda-repository-pin-600
!wget http://developer.download.nvidia.com/compute/cuda/11.0.2/local_installers/cuda-r
!dpkg -i cuda-repo-ubuntu1804-11-0-local_11.0.2-450.51.05-1_amd64.deb
!apt-key add /var/cuda-repo-ubuntu1804-11-0-local/7fa2af80.pub
!apt-get update && sudo apt-get install cuda-toolkit-11-0
!export LD_LIBRARY_PATH=/usr/local/cuda-11.0/lib64:$LD_LIBRARY_PATH
```

Рисунок 2.4 – Встановлення TensorFlow Object Detection API та CUDA

Зображення з набору validation використовуються для перевірки прогресу навчання та коригування гіперпараметрів, таких як швидкість навчання. На відміну від зображень з набору train, ці зображення застосовуються лише періодично під час навчання, наприклад, один раз на певну кількість кроків. Це допомагає виявити можливе перенавчання моделі (overfitting), коли вона добре працює на тренувальних даних, але погано узагальнює на нові дані. Валідаційний набір слугує своєрідним контрольним механізмом, який дозволяє налаштовувати модель для досягнення кращих результатів на невідомих даних.

Набір test використовується для остаточного тестування моделі і визначення її точності на даних, які вона ніколи не бачила під час навчання. Це

ключовий етап, оскільки саме він показує, наскільки добре модель зможе працювати в реальних умовах. Зображення з тестового набору не використовуються під час навчання або валідації, що гарантує незалежну оцінку продуктивності моделі. Якщо модель демонструє високу точність на тестовому наборі, це свідчить про її здатність добре узагальнювати знання і робити точні передбачення на нових, невідомих даних.

Процес розподілу зображень на набори train, validation, та test може бути автоматизований за допомогою скриптів. Наприклад, на зображенні (Рис. 2.5) представлено код на Python, який визначає кількість файлів для кожної категорії та випадковим чином переміщує файли до відповідних папок.

На наступному етапі ми перейдемо до створення карти міток (labelmap) для нашого детектора об’єктів та перетворення зображень у формат файлу даних, відомий як TFRecords. TensorFlow використовує цей формат для навчання своїх моделей. Щоб досягти цього, ми застосуємо сценарії на Python, які автоматично перетворюють наші дані у необхідний формат. Перед тим як запустити ці сценарії, необхідно визначити карту міток для наших класів об’єктів.

Перший крок полягає у створенні файлу "labelmap.txt", який міститиме список класів, які буде розпізнавати наша модель. У цьому файлі кожен клас повинен бути на окремому рядку. Наприклад, якщо ви хочете розпізнавати такі фрукти як яблуко, банан, апельсин, виноград та полуниця, ваш файл виглядатиме наступним чином (Рис. 2.5).

Перший крок полягає у створенні файлу "labelmap.txt", який міститиме список класів, які буде розпізнавати наша модель. У цьому файлі кожен клас повинен бути на окремому рядку. Наприклад, якщо ви хочете розпізнавати такі фрукти як яблуко, банан, апельсин, виноград та полуниця, ваш файл виглядатиме наступним чином (Рис. 2.6).

```
# Визначення кількості файлів для переміщення до кожної папки
train_percent = 0.8  # 80% файлів йде до train
val_percent = 0.1    # 10% йде до validation
test_percent = 0.1   # 10% йде до test
train_num = int(file_num * train_percent)
val_num = int(file_num * val_percent)
test_num = file_num - train_num - val_num
print('Зображень для переміщення в train: %d' % train_num)
print('Зображень для переміщення в validation: %d' % val_num)
print('Зображень для переміщення в test: %d' % test_num)

# Вибір 80% файлів випадковим чином та переміщення їх у папку train
for i in range(train_num):
    move_me = random.choice(file_list)
    fn = move_me.name
    base_fn = move_me.stem
    parent_path = move_me.parent
    xml_fn = base_fn + '.xml'
    os.rename(move_me, train_path + '/' + fn)
    os.rename(os.path.join(parent_path, xml_fn), os.path.join(train_path, xml_fn))
    file_list.remove(move_me)

# Вибір 10% залишкових файлів та переміщення їх у папку validation
for i in range(val_num):
    move_me = random.choice(file_list)
    fn = move_me.name
    base_fn = move_me.stem
    parent_path = move_me.parent
    xml_fn = base_fn + '.xml'
    os.rename(move_me, val_path + '/' + fn)
    os.rename(os.path.join(parent_path, xml_fn), os.path.join(val_path, xml_fn))
    file_list.remove(move_me)

# Переміщення залишкових файлів у папку test
for i in range(test_num):
    move_me = random.choice(file_list)
    fn = move_me.name
    base_fn = move_me.stem
    parent_path = move_me.parent
    xml_fn = base_fn + '.xml'
    os.rename(move_me, test_path + '/' + fn)
    os.rename(os.path.join(parent_path, xml_fn), os.path.join(test_path, xml_fn))
    file_list.remove(move_me)
```

Рисунок 2.5 – Визначання кількість файлів для кожної категорії та переміщення випадковим чином їх до відповідних папок

```
%%bash
cat <<EOF >> /content/labelmap.txt
apple
banana
orange
grape
strawberry
EOF
```

Рисунок 2.6 – Приклад створення labelmap

Цей файл буде використовуватися для зіставлення ідентифікаторів класів з їхніми текстовими описами під час процесу навчання моделі. Важливо правильно визначити всі класи, які ви плануєте використовувати, щоб модель могла коректно навчитися їх розпізнавати.

Далі, ми створимо TFRecord файли для наборів даних train, validation та test. Ці файли міститимуть як зображення, так і відповідні мітки у зручному для TensorFlow форматі. Для цього ми використовуємо спеціальні скрипти на Python. Основним завданням є зчитування вихідних зображень та їхніх анотацій, а потім перетворення цих даних у формат TFRecord.

На прикладі нижче (Рис. 2.7) показано код, який здійснює перетворення XML-файлів з анотаціями у формат CSV, а потім у TFRecord.

Цей скрипт зчитує всі XML-файли з анотаціями у вказаній директорії, перетворює їх у формат CSV, а потім зберігає результат у нових файлах. Після цього необхідно виконати ще один крок (Рис. 2.8) для конвертації цих CSV-файлів у TFRecord, який буде безпосередньо використовуватись для навчання моделі.

```
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET

def xml_to_csv(path):
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text)
                     )
            xml_list.append(value)
    column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df

def main():
    for folder in ['train','validation']:
        image_path = os.path.join(os.getcwd(), ('images/' + folder))
        xml_df = xml_to_csv(image_path)
        xml_df.to_csv(('images/' + folder + '_labels.csv'), index=None)
        print('Successfully converted xml to csv.')

main()
```

Рисунок 2.7 – Приклад перетворення XML-файлів з анотаціями у формат CSV, а потім у TFRecord

```
!python3 create_csv.py
!python3 create_tfrecord.py --csv_input=images/train_labels.csv --labelmap=labelmap.txt \
    --image_dir=images/train --output_path=train.tfrecord
!python3 create_tfrecord.py --csv_input=images/validation_labels.csv --labelmap=labelmap.txt \
    --image_dir=images/validation --output_path=val.tfrecord
```

Рисунок 2.8 – Конвертація CSV-файлів у TFRecord

Далі ми налаштуємо модель та конфігурацію навчання для нашої моделі виявлення об’єктів. Важливим аспектом є вибір попередньо навченої моделі з

TensorFlow 2 Object Detection Model Zoo, а також налаштування параметрів навчання, таких як швидкість навчання та загальна кількість кроків.

Перш за все, ми визначимо, яку модель будемо використовувати. Для цього встановимо змінну chosen_model на відповідну модель. У прикладі нижче (Рис. 2.9) ми використовуємо популярну модель "ssd-mobilenet-v2-fpnlite-320". Це одна з моделей, яка забезпечує гарний баланс між швидкістю та точністю.

```
chosen_model = 'ssd-mobilenet-v2-fpnlite-320'

MODELS_CONFIG = {
    'ssd-mobilenet-v2': {
        'model_name': 'ssd_mobilenet_v2_320x320_coco17_tpu-8',
        'base_pipeline_file': 'ssd_mobilenet_v2_320x320_coco17_tpu-8.config',
        'pretrained_checkpoint': 'ssd_mobilenet_v2_320x320_coco17_tpu-8.tar.gz',
    },
    'efficientdet-d0': {
        'model_name': 'efficientdet_d0_coco17_tpu-32',
        'base_pipeline_file': 'ssd_efficientdet_d0_512x512_coco17_tpu-8.config',
        'pretrained_checkpoint': 'efficientdet_d0_coco17_tpu-32.tar.gz',
    },
    'ssd-mobilenet-v2-fpnlite-320': {
        'model_name': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8',
        'base_pipeline_file': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.config',
        'pretrained_checkpoint': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz',
    },
}

model_name = MODELS_CONFIG[chosen_model]['model_name']
pretrained_checkpoint = MODELS_CONFIG[chosen_model]['pretrained_checkpoint']
base_pipeline_file = MODELS_CONFIG[chosen_model]['base_pipeline_file']
```

Рисунок 2.9 – Приклад вибору попередньо навченої моделі

Після вибору та завантаження моделі, нам необхідно налаштувати файл конфігурації для цієї моделі (Рис. 2.10). Цей файл містить важливі параметри, такі як розташування файлів зображень та міток, шляхи до контрольних точок попередньо навчених моделей, швидкість навчання, кількість кроків навчання тощо. Для цього ми використаємо базовий файл конфігурації, який надається разом з обраною моделлю, і внесемо до нього необхідні зміни.

Наступні змінні використовуються для контролю кроків навчання:

– num_steps: загальна кількість кроків для навчання моделі. Гарне число для початку — 10 000 кроків. Можливо використовувати і більше кроків. Чим більше кроків, тим довше часу займе тренування;

– batch_size: кількість зображень для використання на кроці навчання. Більший розмір партії дозволяє навчити модель за меншу кількість кроків, але розмір обмежений пам'яттю GPU, доступною для навчання. З графічними процесорами, які використовуються в Colab, 16 є хорошим числом для моделей SSD, а 4 є хорошим для моделей EfficientDet.

```
# Встановлення параметрів тренування для моделі
num_steps = 10000

if chosen_model == 'efficientdet-d0':
    batch_size = 4
else:
    batch_size = 16
```

```
# Встановлення розташування файлів та отримання кількості класів для конфігураційного файлу
pipeline_fname = '/content/models/mymodel/' + base_pipeline_file
fine_tune_checkpoint = '/content/models/mymodel/' + model_name + '/checkpoint/ckpt-0'

def get_num_classes(pbtxt_fname):
    from object_detection.utils import label_map_util
    label_map = label_map_util.load_labelmap(pbtxt_fname)
    categories = label_map_util.convert_label_map_to_categories(
        label_map, max_num_classes=90, use_display_name=True)
    category_index = label_map_util.create_category_index(categories)
    return len(category_index.keys())

num_classes = get_num_classes(label_map_pbtxt_fname)
print('Загальна кількість класів:', num_classes)
```

Рисунок 2.10 – Налаштування файлу конфігурації моделі

Наступним кодом (Рис. 2.11) ми перепишемо файл конфігурації, щоб використовувати налаштовані параметри навчання. Це необхідно для того, щоб забезпечити правильне навчання нашої моделі. Ми використаємо Python для автоматичної заміни необхідних параметрів у завантаженому файлі .config та збережемо його як наш спеціальний файл pipeline_file.config.

Спочатку, використовуючи регулярні вирази, ми замінимо значення таких

параметрів як fine_tune_checkpoint, шляхи до TFRecord файлів для навчальних і тестових датасетів, шлях до файлу з картою міток (label_map), batch_size, кількість кроків навчання (num_steps), кількість класів (num_classes), та тип контрольної точки з "classification" на "detection".

```
%cd /content/models/mymodel
print('запис власного конфігураційного файлу')

with open(pipeline_fname) as f:
    s = f.read()
with open('pipeline_file.config', 'w') as f:

    # Встановлення шляху до fine_tune_checkpoint
    s = re.sub('fine_tune_checkpoint: ".*?"', 'fine_tune_checkpoint: "{}"'.format(fine_tune_checkpoint), s)

    # Встановлення tfrecord файлів для навчальних і тестових датасетів
    s = re.sub('(input_path: ".*?)(PATH_TO_BE_CONFIGURED/train)(.*?")', 'input_path: "{}"'.format(train_record_fname), s)
    s = re.sub('(input_path: ".*?)(PATH_TO_BE_CONFIGURED/val)(.*?")', 'input_path: "{}"'.format(val_record_fname), s)

    # Встановлення шляху до label_map
    s = re.sub('label_map_path: ".*?"', 'label_map_path: "{}"'.format(label_map_pbtxt_fname), s)

    # Встановлення batch_size
    s = re.sub('batch_size: [0-9]+', 'batch_size: {}'.format(batch_size), s)

    # Встановлення кількості кроків тренування, num_steps
    s = re.sub('num_steps: [0-9]+', 'num_steps: {}'.format(num_steps), s)

    # Встановлення кількості класів, num_classes
    s = re.sub('num_classes: [0-9]+', 'num_classes: {}'.format(num_classes), s)

    # Зміна типу контрольної точки з "classification" на "detection"
    s = re.sub('fine_tune_checkpoint_type: "classification"', 'fine_tune_checkpoint_type: "{}"'.format('detection'), s)

    if chosen_model == 'ssd-mobilenet-v2':
        s = re.sub('learning_rate_base: .8', 'learning_rate_base: .08', s)
        s = re.sub('warmup_learning_rate: 0.13333', 'warmup_learning_rate: .026666', s)

    if chosen_model == 'efficientdet-d0':
        s = re.sub('keep_aspect_ratio_resizer', 'fixed_shape_resizer', s)
        s = re.sub('pad_to_max_dimension: true', '', s)
        s = re.sub('min_dimension', 'height', s)
        s = re.sub('max_dimension', 'width', s)

    f.write(s)
```

Рисунок 2.11 – Переписування файлу конфігурації, щоб використовувати налаштовані параметри навчання

Навчання моделі (Рис. 2.12) виконується за допомогою сценарію model_main_tf2.py з TF Object Detection API. Навчання може зайняти різний час, залежно від обраної моделі, розміру партії та кількості кроків навчання. Ми вже визначили всі необхідні параметри та аргументи для model_main_tf2.py у попередніх розділах.

```
# Запуск тренування!
!python /content/models/research/object_detection/model_main_tf2.py \
    --pipeline_config_path={pipeline_file} \
    --model_dir={model_dir} \
    --alsologtostderr \
    --num_train_steps={num_steps} \
    --sample_1_of_n_eval_examples=1
```

Рисунок 2.12 – Запуск тренування моделі

Наша модель повністю навчена та готова до використання для виявлення об’єктів. Наступним кроком є експорт графіка моделі (файл, який містить інформацію про архітектуру та ваги) у формат, сумісний із TensorFlow Lite (Рис. 2.13). Це дозволить використовувати модель на мобільних пристроях та вбудованих системах, де потрібна ефективність та невеликий розмір файлів. Для цього ми скористаємося сценарієм export_tflite_graph_tf2.py.

```
!mkdir /content/custom_model_lite
output_directory = '/content/custom_model_lite'
last_model_path = '/content/training'

!python /content/models/research/object_detection/export_tflite_graph_tf2.py \
    --trained_checkpoint_dir {last_model_path} \
    --output_directory {output_directory} \
    --pipeline_config_path {pipeline_file}
```

Рисунок 2.13 – Експорт графіка моделі у формат, сумісний із TensorFlow Lite

Після експорту графіка моделі ми виконаємо наступний крок (Рис. 2.14): використаємо модуль TFLiteConverter з TensorFlow, щоб перетворити експортований графік у формат .tflite FlatBuffer.

```
# Конвертування експортованого графічного файлу в файл моделі TFLite
import tensorflow as tf

converter = tf.lite.TFLiteConverter.from_saved_model('/content/custom_model_lite/saved_model')
tflite_model = converter.convert()

with open('/content/custom_model_lite/detect.tflite', 'wb') as f:
    f.write(tflite_model)
```

Рисунок 2.14 – Перетворення експортованого графіка у формат .tflite FlatBuffer

Наша кастомна модель, перетворена в формат TFLite, є ключовою частиною нашого дослідження. Однак питання її ефективності виявлення об’єктів на нових зображеннях залишається відкритим. Для оцінки її роботи ми використовуємо набір тестових зображень, які не були використані під час навчання моделі. Це важливо, оскільки дозволяє переконатися, що модель виявляє об’єкти так само ефективно на нових даних, як і на тих, на яких вона тренувалася.

Наведена частина коду (Рис. 2.15) реалізує процес завантаження зображень, ініціалізації моделі TensorFlow Lite та виконання виявлення об’єктів. Спочатку код здійснює пошук імен файлів усіх зображень у вказаній тестовій папці за допомогою функції glob, яка шукає файли з розширеннями .jpg, .JPG, .png та .bmp. Потім завантажується карта міток, що зберігається у текстовому файлі, де кожен рядок відповідає певній категорії об’єктів.

Кожне зображення аналізується на предмет виявлення об’єктів. Якщо впевненість виявлення перевищує мінімальний поріг, малюються прямокутники навколо виявлених об’єктів та відповідні мітки з назвою об’єкта та відсотком впевненості. Це дозволяє візуально оцінити ефективність моделі.

Після візуалізації результати можуть бути збережені у вигляді текстових файлів для подальшого аналізу та обчислення метрик, таких як середній показник точності визначення об’єктів (mAP). Це допомагає отримати кількісну оцінку ефективності моделі на нових даних.

```
import os, cv2, sys, glob, random
import numpy as np
from tensorflow.lite.python.interpreter import Interpreter
import matplotlib.pyplot as plt
%matplotlib inline

def tflite_detect_images(modelpath, imgpath, lblpath, min_conf=0.5, num_test_images=10,
                         savepath='/content/results', txt_only=False):

    images = glob.glob(imgpath + '/*.jpg') + glob.glob(imgpath + '/*.JPG') + \
             glob.glob(imgpath + '/*.png') + glob.glob(imgpath + '/*.bmp')

    with open(lblpath, 'r') as f:
        labels = [line.strip() for line in f.readlines()]

    interpreter = Interpreter(model_path=modelpath)
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    height, width = input_details[0]['shape'][1], input_details[0]['shape'][2]
    float_input = (input_details[0]['dtype'] == np.float32)
    input_mean, input_std = 127.5, 127.5

    images_to_test = random.sample(images, num_test_images)

    for image_path in images_to_test:
        image = cv2.imread(image_path)
        image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        imH, imW, _ = image.shape
        image_resized = cv2.resize(image_rgb, (width, height))
        input_data = np.expand_dims(image_resized, axis=0)

        if float_input:
            input_data = (np.float32(input_data) - input_mean) / input_std

        interpreter.set_tensor(input_details[0]['index'], input_data)
        interpreter.invoke()

        boxes = interpreter.get_tensor(output_details[1]['index'])[0]
        classes = interpreter.get_tensor(output_details[3]['index'])[0]
        scores = interpreter.get_tensor(output_details[0]['index'])[0]

        detections = []
```

Рисунок 2.15 – Процес завантаження зображень, ініціалізації моделі TensorFlow Lite та виконання виявлення об’єктів

Наступний блок коду (Рис. 2.16) встановлює шляхи до тестових зображень моделі та викликає функцію тестування для оцінки роботи кастомної моделі,

перетвореної у формат TFLite. Шлях до тестових зображень PATH_TO_IMAGES вказує на теку з зображеннями, які будуть використовуватись для перевірки моделі. Модель PATH_TO_MODEL зчитується з файлу .tflite, та PATH_TO_LABELS містить карту міток, необхідних для інтерпретації результатів виявлення об’єктів (Рис. 2.17).

```
# Налаштування змінних для запуску моделі користувача
PATH_TO_IMAGES='/content/images/test'
PATH_TO_MODEL='/content/custom_model_lite/detect.tflite'
PATH_TO_LABELS='/content/labelmap.txt'
min_conf_threshold=0.5
images_to_test = 10

# Запуск тестування
tflite_detect_images(PATH_TO_MODEL, PATH_TO_IMAGES, PATH_TO_LABELS, min_conf_threshold, images_to_test)
```

Рисунок 2.16 – Встановлення шляхів до тестових зображень моделі та виклик функції тестування

Рисунок 2.17 – Приклад виведення результатів виявлення об’єктів

Тепер ми маємо візуальне уявлення про те, як наша модель працює на тестових зображеннях, але як ми можемо кількісно виміряти її точність?

Одним із популярних методів вимірювання точності моделі виявлення об’єктів є «середня точність» (mAP). Загалом, що вищий показник mAP, то краще ваша модель виявляє об’єкти на зображеннях.

Ми використаємо інструмент калькулятора mAP на GitHub, щоб визначити оцінку mAP нашої моделі. Спочатку нам потрібно клонувати репозиторій і видалити наявні приклади даних . Ми також завантажимо сценарій для взаємодії з калькулятором (Рис 2.18).

```
%%bash
git clone https://github.com/Cartucho/mAP \
    /content/mAP
cd /content/mAP
rm input/detection-results/*
rm input/ground-truth/*
rm input/images-optional/*
wget https://raw.githubusercontent.com/EdjeElectronics/\
TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/\
master/util_scripts/calculate_map_cartucho.py
```

Рисунок 2.18 – Встановлення калькулятора mAP

Далі ми скопіюємо зображення та анотаційні дані з папки test у відповідні папки всередині клонованого сховища. Вони використовуватимуться як «основні дані правдивості», з якими порівнюватимуться результати виявлення нашої моделі. Інструмент калькулятор очікує даних анотації у форматі, який відрізняється від формату файлу Pascal VOC .xml, який ми використовуємо. На щастя, він надає простий сценарій, convert_gt_xml.py, для перетворення в очікуваний формат .txt (Рис. 2.19).

```
!cp /content/images/test/* /content/mAP/input/images-optional
!mv /content/mAP/input/images-optional/*.xml /content/mAP/input/ground-truth/
!python /content/mAP/scripts/extra/convert_gt_xml.py
```

Рисунок 2.19 – Копіювання та перетворення анотаційних даних в потрібний формат

Ми налаштували базові дані, але тепер нам потрібні фактичні результати виявлення з нашої моделі. Результати виявлення порівнюватимуться з правдивими даними, щоб обчислити точність моделі в mAP.

Функцію перевірки, яку ми визначили раніше, можна використовувати для створення даних виявлення для всіх зображень у папці test. Ми будемо використовувати її так само, як і раніше, за винятком того, що цього разу ми будемо зберігати результати виявлення в папці detection-results.

Для цього нам потрібно змінити шляхи до відповідних папок. Шлях до тестових зображень PATH_TO_IMAGES вказує на теку з зображеннями, які будуть використовуватись для перевірки моделі. Шлях до моделі PATH_TO_MODEL вказує на файл .tflite, що містить нашу модель, а шлях до міток PATH_TO_LABELS вказує на файл, що містить карту міток для інтерпретації результатів. Шлях до папки для збереження результатів виявлення PATH_TO_RESULTS вказує на теку detection-results у клонованому репозиторії mAP.

Ми також встановлюємо мінімальний поріг впевненості min_conf_threshold на рівні 0.1 для більшої гнучкості в оцінці виявлених об’єктів. Потім ми створюємо список зображень для тестування, обмежуючи їх кількість до 500 або менше, залежно від наявної кількості зображень.

Наступний крок - викликати функцію tflite_detect_images для виконання виявлення об’єктів на тестових зображеннях та збереження результатів у текстових файлах у папці detection-results.

Після того, як ми отримали результати виявлення, ми переходимо до каталогу mAP і запускаємо сценарій для обчислення mAP з використанням нашої карти міток.

Цей процес дозволяє нам не лише візуально оцінити продуктивність нашої моделі, але й отримати кількісні показники (Рис. 2.20).

```
PATH_TO_IMAGES = '/content/images/test'
PATH_TO_MODEL = '/content/custom_model_lite/detect.tflite'
PATH_TO_LABELS = '/content/labelmap.txt'
PATH_TO_RESULTS = '/content/mAP/input/detection-results'
min_conf_threshold = 0.1

image_list = (glob.glob(PATH_TO_IMAGES + '/*.jpg') +
              glob.glob(PATH_TO_IMAGES + '/*.JPG') +
              glob.glob(PATH_TO_IMAGES + '/*.png') +
              glob.glob(PATH_TO_IMAGES + '/*.bmp'))
images_to_test = min(500, len(image_list))

txt_only = True

print('Починається перевірка на %d зображеннях...' % images_to_test)
tflite_detect_images(PATH_TO_MODEL, PATH_TO_IMAGES,
                     PATH_TO_LABELS, min_conf_threshold,
                     images_to_test, PATH_TO_RESULTS, txt_only)
print('Перевірку завершено!')

%cd /content/mAP
!python calculate_map_cartucho.py --labels=/content/labelmap.txt
```

Рисунок 2.20 – Обчислення mAP

Тепер, коли наша кастомна модель навчена та перетворена у формат TFLite, її можна завантажити та використати в програмі.

Наступний блок коду (Рис. 2.21), копіює файли карти міток у папку моделі, заархівовує її в папку zip, а потім завантажує. Папка zip буде містити модель detect.tflite і файли карти міток labelmap.txt, які необхідні для запуску моделі.

```
!cp /content/labelmap.txt /content/custom_model_lite
!cp /content/labelmap.pbtxt /content/custom_model_lite
!cp /content/models/mymodel/pipeline_file.config /content/custom_model_lite

%cd /content
!zip -r custom_model_lite.zip custom_model_lite

from google.colab import files

files.download('/content/custom_model_lite.zip')
```

Рисунок 2.21 – Завантаження натренованої моделі

2.6. Тестування та налагодження програмної розробки

Тестування програмного засобу – це критичний етап розробки, який має на меті перевірку відповідності програмного продукту заданим вимогам та виявлення можливих дефектів. Воно охоплює проведення аналізу компонентів системи за допомогою як ручних, так і автоматизованих інструментів. Основною метою тестування є ідентифікація помилок, дефектів або невідповідностей, що можуть виникнути в процесі розробки.

Під час тестування нашого інструменту для тренування моделей TensorFlow Object Detection в Google Colab були виявлені та усунені кілька типових проблем. Серед них архітектурні недоліки, які впливали на загальну стабільність системи, та декілька функціональних помилок, які перешкоджали коректній роботі програми.

Тестування функціональних вимог показало, що користувачі можуть безперешкодно виконувати всі необхідні операції, такі як завантаження зображень, тренування моделей та збереження результатів. Цей етап був проведений на різних операційних системах, включаючи Linux, Windows 7 та Windows 10, що підтвердило сумісність та стабільність роботи програмного засобу на різних платформах.

Ручне тестування також продемонструвало гарні результати. У підсумку, наш інструмент виявився надійним та ефективним засобом для тренування моделей виявленя об’єктів.

2.7. Рекомендації по використанню та впровадженню програмного засобу для тренування моделей виявлення об’єктів

Для успішного впровадження інструменту у робочий процес важливо врахувати кілька ключових аспектів. По-перше, необхідно забезпечити відповідну апаратну інфраструктуру. Тренування моделей обробки зображень є ресурсомістким процесом, тому рекомендується використовувати потужні

графічні процесори (GPU) для прискорення навчання. Високопродуктивні сервери або хмарні обчислювальні ресурси можуть значно скоротити час тренування і підвищити ефективність роботи.

По-друге, потрібно підготувати відповідний набір даних для тренування моделі. Якість та обсяг даних мають вирішальне значення для продуктивності моделі. Рекомендується збирати великий та різноманітний набір зображень, які охоплюють всі можливі варіанти об’єктів, що виявлятимуться. Крім того, важливо ретельно анотувати зображення, щоб забезпечити точність міток.

Після налаштування інфраструктури та підготовки даних можна розпочати тренування моделей. Інструмент для тренування моделей TensorFlow Object Detection автоматизує більшість процесів, що значно спрощує роботу з моделями. Під час тренування важливо стежити за метриками продуктивності, такими як точність, втрата і час тренування. Це допоможе виявити можливі проблеми на ранніх етапах і внести необхідні корективи.

Інструмент також надає можливість тестування натренованих моделей на тестовому наборі зображеннях. Це дозволяє оцінити, наскільки добре модель справляється з новими даними, які вона не бачила під час тренування.

Одним з ключових аспектів успішного використання інструменту є постійне покращення та оптимізація моделі. Для цього можна використовувати кілька підходів. По-перше, варто експериментувати з різними архітектурами моделей та гіперпараметрами. Наприклад, використання моделей EfficientDet може значно підвищити точність виявлення об’єктів за рахунок більш ефективного використання обчислювальних ресурсів.

По-друге, можна застосовувати техніки аугментації даних для збільшення різноманітності тренувального набору. Це може включати різні перетворення зображень, такі як обертання, зміна масштабу, кольору тощо. Аугментація допомагає моделі краще узагальнювати інформацію і справлятися з новими, незнайомими зображеннями.

ВИСНОВКИ

Проведене дослідження та аналіз існуючих інструментів для тренування моделей виявлення об’єктів дозволило отримати глибше розуміння сучасних підходів у цій області. Виявлення об’єктів є важливою складовою сучасних систем комп’ютерного зору, що знаходить застосування в різних сферах від промисловості до наукових досліджень.

Розроблений інструмент на базі TensorFlow успішно впроваджено у середовищі Google Colab, що забезпечує зручність та доступність для широкого кола користувачів. Його основна функціональність полягає у тренуванні кастомних моделей виявлення об’єктів, що дає можливість адаптувати модель до специфічних вимог задачі.

В ході розробки було реалізовано можливість тестування моделі на реальних даних з використанням метрики середньої точності (mAP), що дозволило оцінити її ефективність. Високий показник mAP свідчить про високу точність моделі у виявленні об’єктів на зображеннях, що є важливим критерієм для її успішного використання у практичних застосуваннях.

Додавання функціоналу для завантаження натренованих моделей на пристрої робить розроблений інструмент більш гнучким та зручним у використанні для кінцевих користувачів. Це дозволяє з легкістю впроваджувати результати тренування у реальних умовах і використовувати моделі для вирішення конкретних завдань.

У підсумку, розроблений інструмент для тренування моделей TensorFlow Object Detection у Google Colab є потужним інструментом, який поєднує в собі сучасні технології машинного навчання з високою доступністю і зручністю використання. Подальші можливості його вдосконалення включають розширення функціональності та оптимізацію для швидкого розгортання у різних сценаріях застосування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке нейромережі? Все про нейронні мережі. *INCRYPTED*. URL: https://incrypted.com/ua/shcho-take nejromerezhi/ (дата звернення: 28.05.2024).
2. Довідник по machine learning – recurrent neural network | база знань IT. *База знань IT технологій*. URL: https://itwiki.dev/data-science/ml-reference/ml-glossary/recurrent-neural-network (дата звернення: 28.05.2024).
3. Розробка і навчання нейронної мережі. Deep learning Neural network. *Evergreen - web розробка і діджиталізація бізнесу за допомогою AI продуктів*. URL: https://evergreens.com.ua/ua/development-services/neural-network.html (дата звернення: 28.05.2024).
4. Типові архітектури нейронних мереж. *StudFiles*. URL: https://studfile.net/preview/5740125/page:4/ (дата звернення: 28.05.2024).
5. Neural Networks in Materials Science. *Phase Transformations and Complex Properties*. URL: https://www.phase-trans.msm.cam.ac.uk/abstracts/neural.review.pdf (дата звернення: 29.05.2024).
6. Що таке багатошаровий перцептрон (Multilayer Perceptron, MLP) у машинному навчанні? | TheTransmitted. *TheTransmitted*. URL: https://thetransmitted.com/adlucem/shho-take-mlp-u-mashynnomu-navchanni/ (дата звернення: 28.05.2024).
7. 8 common types of neural networks. *Coursera*. URL: https://www.coursera.org/in/articles/types-of-neural-networks (дата звернення: 28.05.2024).
8. Ajith Abraham Cyber Web Shack. URL: https://wsc10.softcomputing.net/ann_chapter.pdf (дата звернення: 28.05.2024).
9. Ashtari H. What is a neural network and its types?-. *Spiceworks Inc*. URL: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-a-neural-network/ (дата звернення: 28.05.2024).

10. AutoML Google і нові можливості машинного навчання. *Evergreen - web розробка і діджиталізація бізнесу за допомогою AI продуктів*. URL: https://evergreens.com.ua/ua/articles/automl.html (дата звернення: 29.05.2024).
11. What is amazon rekognition? - amazon rekognition. URL: https://docs.aws.amazon.com/rekognition/latest/dg/what-is.html (дата звернення: 29.05.2024).
12. Global Vision Press. URL: https://gvpress.com/journals/IJHIT/vol9_no11/34.pdf (дата звернення: 28.05.2024).
13. Google colab або google colaboratory: що це таке. *Hardware libre*. URL: https://www.hwlibre.com/uk/google-colaboratory/ (дата звернення: 29.05.2024).
14. What is python for machine learning?. *Noble Desktop. URL: https://www.nobledesktop.com/learn/python-for-machine-learning/what-is-python-for-machine-learning (дата звернення: 29.05.2024).*
15. *Recurrent Convolutional Neural Network for Object Recognition*. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Liang_Recurrent_Convolutional_Neural_2015_CVPR_paper.pdf (дата звернення: 28.05.2024).
16. Tensorflow що це і які основні можливості застосування. *FoxmindEd*. URL: https://foxminded.ua/tensorflow-shcho-tse/ (дата звернення: 28.05.2024).
17. Types of neural networks and definition of neural network. *Great Learning Blog.* URL: https://www.mygreatlearning.com/blog/types-of-neural-networks/ (дата звернення: 28.05.2024).
18. What are convolutional neural networks? | IBM. *IBM - United States*. URL: https://www.ibm.com/topics/convolutional-neural-networks (дата звернення: 28.05.2024).
19. What is an autoencoder? | IBM. *IBM - United States*. URL: https://www.ibm.com/topics/autoencoder (дата звернення: 28.05.2024).

ДОДАТОК А
ТЕХНІЧНЕ ЗАВДАННЯ

Вступ

Технічне завдання описує інструмент для тренування кастомних моделей виявлення об’єктів за допомогою TensorFlow. Проєкт втілює програмний засіб, що надає можливість ефективно створювати, навчати та тестувати моделі машинного навчання для виявлення об’єктів на зображеннях.

Підстави для розробки

Розробка проводиться на основі індивідуального завдання, поставленого керівником бакалаврської роботи для спеціальності 122 “Комп’ютерні науки”.

Призначення розробки

Метою розробки є створення інструменту для тренування та тестування кастомних моделей виявлення об’єктів з використанням TensorFlow у середовищі Google Colab.

Вимоги до функціональних характеристик розробки:

– система повинна забезпечувати користувачам можливість завантаження власних наборів даних для тренування моделей виявлення об’єктів у середовищі;

– повинна бути передбачена функціональність для налаштування параметрів тренування моделей, включаючи вибір архітектури моделі, параметри навчання та критерії зупинки.

– повинен бути забезпечений функціонал для збереження та завантаження моделей;

Вимоги до надійності: система повинна працювати без збоїв та затримок.

Умови експлуатації: веб-застосунок має бути доступним для користувачів цілодобово.

Вимоги до інформаційної і програмної сумісності: платформа повинна бути сумісною з різними веб-переглядачами та працювати на різних операційних системах.

Програмний засіб повинен працювати в режимі онлайн і бути доступним для користувачів на пристроях з надійним інтернет-з’єднанням.

Вимоги до програмної документації

До складу документів входять дипломна робота та технічне завдання.

Стадії і етапи розробки:

– збір і позначення зображень;

– встановлення TensorFlow Object Detection Dependencies;

– обробка та підготовка даних;

– налаштування параметрів навчання моделі, таких як кількість кроків, розмір партії, швидкість навчання тощо;

– запуск процесу тренування моделі для виявлення об’єктів за допомогою TensorFlow Lite;

– перетворення натренованої моделі у формат TensorFlow Lite для оптимізації використання на мобільних та вбудованих пристроях;

– перевірка моделі TensorFlow Lite та обчислення середньої точності (mAP) для визначення ефективності виявлення об’єктів;

– завантаження та збереження кінцевої моделі TensorFlow Lite для подальшого використання в застосунках.

Порядок контролю і приймання

Приймання результатів роботи здійснюється в процесі здачі бакалаврської роботи.

ДОДАТОК Б
ІНСТРУКЦІЯ КОРИСТУВАЧУ

Загальні відомості

Інструмент для тренування моделей TensorFlow Object Detection в Google Colab – це програмний засіб, розроблений для спрощення процесу підготовки та навчання моделей виявлення об’єктів.

Функціональне призначення

Інструмент дозволяє користувачу завантажувати дані, налаштовувати параметри моделі, проводити тренування та валідацію моделей об’єктного виявлення на базі TensorFlow. Інструмент також підтримує збереження та експорт навченої моделі для подальшого використання.

Опис роботи програми

Для початку роботи з інструментом користувачу необхідно увійти до свого облікового запису Google та відкрити Google Colab. Після цього можна завантажити файл Jupyter Notebook, що містить всі необхідні кроки для підготовки та тренування моделі.

Користувач починає з налаштування середовища, яке включає встановлення необхідних бібліотек та завантаження даних. Потім необхідно налаштувати параметри моделі, такі як архітектура, гіперпараметри та інші опції, що впливають на процес навчання. Далі користувач завантажує датасет з зображеннями та анотаційними файлами, що будуть використовуватися для тренування та тестування моделі.

Після підготовки даних можна розпочати тренування моделі. Інструмент автоматично обробляє дані, запускає процес навчання. Після завершення тренування користувач може випробувати модель на тестовому наборі даних та обчислити середню точності (mAP), щоб оцінити її точність та ефективність.

Для збереження результатів тренування інструмент пропонує функції експорту навченої моделі, яку можна використовувати в інших проєктах або для подальшого вдосконалення.

АНОТАЦІЯ

Ковальчук В.В. Розробка інструменту для тренування моделей TensorFlow Object Detection. Рукопис.

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 122 Комп'ютерні науки. Волинський національний університет імені Лесі Українки, Луцьк, 2024 р.

Ця кваліфікаційна робота присвячена розробці інструменту для тренування моделей виявлення об'єктів на основі платформи TensorFlow. Враховуючи постійне зростання обсягів даних та потребу у швидкому і точному аналізі зображень, розробка таких інструментів є актуальною.

У першому розділі досліджено основні поняття про нейронні мережі, їхню архітектуру, методи навчання та типи нейронних мереж. Особлива увага приділена функціональним можливостям TensorFlow та аналізу інструментів, таких як Google Cloud AutoML та Amazon Rekognition.

Другий розділ присвячено розробці та реалізації інструменту для тренування кастомних моделей виявлення об'єктів. Описано постановку задачі, загальну структуру проєкту, вибір моделі розробки та обґрунтування вибору інструментальних засобів, включаючи TensorFlow, Python та середовище розробки Google Colab. Розглянуто особливості програмної реалізації, тестування та налагодження програмного засобу.

Результати дослідження показали, що розроблений інструмент успішно впроваджено у середовищі Google Colab, що забезпечує зручність та доступність для широкого кола користувачів. Інструмент дозволяє тренувати кастомні моделі виявлення об'єктів та оцінювати їх ефективність за допомогою метрики середньої точності (mAP). Високий показник mAP підтверджує високу точність моделі у виявленні об'єктів на зображеннях.

Ключові слова: TensorFlow, виявлення об'єктів, нейронні мережі, машинне навчання, Google Colab, розпізнавання зображень, навчання моделей.