

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Метод розпізнавання продуктів для визначення наявності за
допомогою нейронної мережі»**

Виконав студент групи ІПЗм-22-2
ПРЕДЧУК Тарас Віталійович

Керівник роботи:
КРАВЧЕНКО Світлана Миколаївна

Рецензент:
ВАКАЛЮК Тетяна Анатоліївна

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«18» вересня 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: ПРЕДЧУК Тарас Віталійович

Керівник роботи: КРАВЧЕНКО Світлана Миколаївна

Тема роботи: «Метод розпізнавання продуктів для визначення наявності за допомогою нейронної мережі», затверджена Наказом закладу вищої освіти від «18» вересня 2023 р. № 479/с

Вихідні дані для роботи: предметом досліджень є засоби розробки сучасних систем комп'ютерного зору, що використовують методи класифікації машинного навчання. Об'єктом досліджень є моделі та методи класифікації машинного навчання.

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Кравченко С.М.	22.09.2023р.	22.09.2023р.
2	Кравченко С.М.	22.09.2023р.	22.09.2023р.
3	Кравченко С.М.	22.09.2023р.	22.09.2023р.

РЕФЕРАТ

Кваліфікаційна робота присвячена розробці методу розпізнавання продуктів для визначення наявності за допомогою нейронної мережі.

Завдання полягає у аналізі методів та архітектур нейронних мереж для розпізнавання об'єктів. Кваліфікаційна робота складається з програмного комплексу, який включає в себе нейронну мережу, серверну частину, а також веб-інтерфейс та пояснювальної записки. Пояснювальна записка складається з вступу, трьох розділів, висновків, списку використаних джерел, який складається з 27 найменувань, 62 рисунків, 2 таблиць та 3 додатків.

В першому розділі було проаналізовано предметну область. Здійснено аналіз необхідних бібліотек, методів машинного навчання та архітектур нейронних мереж. Сформовано завдання на розробку.

В другому розділі було проаналізовано вхідні дані. Розроблено та навчено нейронну мережу. Створено структуру та розроблено веб-додаток, який складається з серверної та клієнтської частиною.

В третьому розділі здійснено навчання системи комп'ютерного зору до отримання максимального результату та порівняно основні вихідні показники навчання. Протестовано нейронну мережу у вигляді веб-додатку та описано результат.

Висновок містить підсумок проведеного дослідження.

Ключові слова: НЕЙРОННА МЕРЕЖА, КОМП'ЮТЕРНИЙ ЗІР, PYTHON, МАШИННЕ НАВЧАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ, YOLO.

ABSTRACT

The qualification work is devoted to the development of a product recognition method for determining availability by using a neural network.

The task is to analyze the methods and architectures of neural networks for object recognition. The qualification work consists of a software package that includes a neural network, a server part, as well as a web interface and an explanatory note. The explanatory note consists of an introduction, three sections, conclusions, a list of references , which consists of 27 titles, 62 Figures, 2 tables and 3 appendices.

In the first chapter, the subject area was analyzed. The necessary libraries, the methods of machine learning and the architectures of neural network were analyzed. A development task has been created.

In the second section, the input data was analyzed. A neural network has been developed and trained. A structure was created and a web application was developed, which consists of a server and client part.

In the third chapter, the computer vision system was trained to get the maximum result and the main initial training indicators were compared. A neural network in the form of a web application was tested and the result was described.

The conclusion contains the result of the searching work.

Keywords: NEURAL NETWORK, COMPUTER VISION, PYTHON, MACHINE LEARNING, ARTIFICIAL INTELLIGENCE, YOLO.

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	9
1.1. Аналіз існуючих бібліотек.....	9
1.2. Методи навчання штучного інтелекту	13
1.3. Аналіз архітектур розпізнавання.....	18
1.4. Постановка завдання на розробку.....	25
Висновки до першого розділу	26
РОЗДІЛ 2. РОЗРОБКА СИСТЕМИ ДЕТЕКТУВАННЯ ЗА ДОПОМОГОЮ МОДЕЛІ YOLOV8N.....	27
2.1. Аналіз вхідних даних	27
2.2. Формування набору даних.....	29
2.3. Розробка та навчання моделі	33
2.4. Проектування та розробка веб-додатку.....	38
Висновки до другого розділу	48
РОЗДІЛ 3. ДОСЛІДЖЕННЯ ПРАЦЕЗДАТНОСТІ ДОДАТКУ ТА ЕФЕКТИВНОСТІ РІШЕННЯ ЗАДАЧІ РОЗПІЗНАВАННЯ	49
3.1. Оцінка та аналіз якості моделі.....	49
3.2. Розгортання та тестування моделі у вигляді веб-додатку	63
Висновки до третього розділу	71
ВИСНОВКИ	72
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	74
ДОДАТКИ	77

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

NLP – Natural Language Processing

ML – Machine Learning

CNN – Convolutional Neural Network

VGG – Visual Geometry Group

ResNet – Residual Network

YOLO – You Only Look Once

ВСТУП

Застосування комп'ютерного зору має великий потенціал для поліпшення ефективності, точності та автоматизації різних процесів. Він може допомогти впроваджувати нові рішення, які раніше були неоспівані через відсутність можливостей аналізу зображень. Тому розуміння та використання комп'ютерного зору стає актуальним і необхідним завданням для майбутнього розвитку товарів і послуг.

Актуальність теми. Розпізнавання продуктів за допомогою нейронних мереж дозволяє автоматизувати цей процес, що значно прискорює процес ідентифікації продуктів. Замість ручного сканування та перевірки продукту, можна використовувати нейронну мережу, яка здатна швидко та точно визначити, який продукт перед нею.

Нейронні мережі можуть бути навчені на великих наборах даних для розпізнавання продуктів. Це дозволяє досягти високої точності визначення продуктів на основі їх зображень або характеристик. Точність розпізнавання може перевищувати людські можливості і забезпечувати надійність результатів.

Розпізнавання продуктів за допомогою нейронних мереж може бути застосоване в різних галузях, таких як роздрібна торгівля, логістика, виробництво, медицина та інші. Він може допомогти автоматизувати процеси і пов'язану з ними ідентифікацію продуктів, що полегшує роботу і знижує ймовірність помилок.

Розпізнавання продуктів за допомогою нейронних мереж може допомогти виявити підроблені або контрафактні продукти. Шаблони, навчені нейронною мережею, можуть розпізнати відмінності між оригінальними і підробленими продуктами, що дозволяє швидко виявити шахрайство та забезпечити безпеку споживачам.

Метою дослідження є розробка системи за допомогою нейронної мережі, яка буде розпізнавати наявність продуктів.

Встановлена мета обумовлює наступні завдання:

1. Проведення аналізу існуючих систем комп'ютерного зору;
2. Визначення оптимального способу реалізації системи;
3. Створення датасету з необхідним матеріалом;
4. Розробка системи комп'ютерного зору на основі нейронної мережі;
5. Розробка веб-додатку;
6. Навчання системи комп'ютерного зору до отримання максимального результату;
7. Тестування розробленої системи за допомогою веб-інтерфейсу.

Об'єктом досліджень є моделі та методи класифікації машинного навчання.

Предметом досліджень є засоби розробки сучасних систем комп'ютерного зору, що використовують методи класифікації машинного навчання.

Практичне застосування: Розроблена система комп'ютерного зору може слугувати допомогою середньому та великому бізнесу у сфері мерчендайзингу. При належному розвитку створеного прототипу, може застосовуватися як основний інструмент для розпізнавання продукції, а також для автоматизації процесу перевірки наявності продукції.

Апробація результатів дослідження за тематикою кваліфікаційної роботи доповідалися на VI Всеукраїнській науково-технічній конференції здобувачів вищої освіти і молодих учених «Комп'ютерні технології: інновації, проблеми, рішення», 29-30 листопада 2023 року. Житомир: "Житомирська політехніка", 2023 [26, 27].

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз існуючих бібліотек

Перш за все, визначимося з мовою програмування, якою буде реалізовано нейронну мережу. Найпопулярнішою мовою для роботи у сфері ML є Python.

Популярність Python обумовлена його універсальністю та гнучкістю, що дозволяє комбінувати його з іншими мовами програмування, такими як Java та C#. Крім того, Python практично використовується на всіх операційних системах і платформах.

Мова має широкий вибір бібліотек для машинного навчання та штучного інтелекту, що дає розробникам можливість вибрати потрібну для вирішення конкретних завдань. Завдяки об'єктно-орієнтованому підходу Python добре підходить для ефективної обробки даних [1].

Це лише деякі аспекти, що роблять Python однією з кращих мов для машинного та глибокого навчання та штучного інтелекту:

- Безкоштовність і відкритий код сприяють спільноті та забезпечують постійне вдосконалення.
- Розширені бібліотеки гарантують рішення для різноманітних проблем.
- Легкість використання й інтеграції робить його доступним для користувачів на різних рівнях навиків.
- Підвищує продуктивність через зменшення часу на написання коду та відладку.
- Застосовується для обчислення та обробки природної мови.
- Ідеально взаємодіє з модулями коду C і C++.

Визначивши мову програмування розглянемо найпопулярніші бібліотеки цієї мови, які слугують для машинного навчання, а також роботи з великими об'ємами даних.

1. NumPy

Раніше відома як Numeric, NumPy зараз має повністю відкритий код і значну кількість учасників. Цю бібліотеку також визнають найкращою у своєму роді для машинного навчання та штучного інтелекту.

NumPy головним чином використовується дослідниками даних для здійснення різноманітних математичних операцій над об'ємними масивами та матрицями [2].

Масиви NumPy потребують значно менше місця для зберігання, порівняно з іншими списками Python. Вони швидкі та прості у використанні, що робить їх чудовим вибором для підвищення ефективності моделей машинного навчання без зайвих зусиль. Іншою привабливою особливістю є можливість інтеграції коду C, C++ і Fortran в NumPy.

2. SciPy

NumPy настільки популярний, що на його базі розроблено декілька бібліотек, включаючи SciPy. SciPy також є вільною бібліотекою з відкритим кодом.

Ця бібліотека орієнтована на обробку обширних наборів даних та виконання наукових та технічних розрахунків. SciPy має вбудовані модулі для оптимізації масивів та лінійної алгебри, подібно до NumPy, й відіграє вагомую роль у науковому аналізі та розробці.

Це, власне, і є привабливість SciPy: вона використовує всі можливості NumPy, перетворюючи їх на зручні наукові інструменти. SciPy часто використовується для обробки зображень та надає основні функції обробки для високорівневих математичних завдань, що не відносяться до науки.

3. PyTorch

PyTorch, розроблений командою Facebook, є відкритою бібліотекою Python для машинного навчання, що ґрунтується на фреймворку мови програмування C, відомому як Torch. Це дає PyTorch можливість мати інтерфейс C++ для підтримки C++. Фреймворк PyTorch вважається одним із лідерів у сфері фреймворків для машинного та глибокого навчання.

Ця бібліотека має велику кількість інструментів для обробки даних і може легко інтегруватися з іншими Python-бібліотеками, такими як NumPy. PyTorch генерує обчислювальні графи, які можна змінювати у процесі виконання програм.

Особливо вдало PyTorch використовується у проектах з машинного та глибокого навчання, таких як обробка природної мови (NLP) та комп'ютерний зір.

Однією з ключових особливостей, що виділяє PyTorch, є його висока швидкодія, яка забезпечується навіть у випадках роботи з складними графіками. PyTorch також демонструє велику гнучкість, успішно функціонуючи як на звичайних, так і на центральних чи графічних процесорах.

Якщо потрібні додаткові функціональні можливості, PyTorch постачається з різноманітними API, які дозволяють розробникам розширювати бібліотеку та використовувати інструменти для обробки природної мови.

4. Scikit-learn

Незважаючи на те, що Scikit-learn стала самостійною бібліотекою Python на Github і отримала визнання від великих компаній, таких як Spotify, вона починала як стороннє розширення для бібліотеки SciPy.

Scikit-learn – це бібліотека з різноманітними можливостями, що використовується для класичних алгоритмів машинного навчання, таких як виявлення спаму, розпізнавання зображень, прогнозування та сегментація клієнтів.

Ця бібліотека легко інтегрується з іншими платформами машинного навчання, такими як NumPy і Pandas. Вона підтримує різноманітні алгоритми, включаючи класифікацію, регресію, кластеризацію та багато інших.

Scikit-learn, проста у використанні та гнучка, є чудовою бібліотекою для моделювання даних. Проте для завдань, пов'язаних із завантаженням, обробкою та візуалізацією даних, існують ще кращі бібліотеки. Тим не менш, Scikit-learn можна використовувати на всіх етапах – від дослідження до розгортання систем машинного навчання [3].

5. Pandas

Pandas – це інструмент для аналізу та обробки даних з відкритим кодом, який володіє високою швидкістю, потужністю, гнучкістю та простотою використання. Ця бібліотека була розроблена AQR Capital Management у 2008 році і стала доступною громадськості наприкінці 2009 року. Pandas ідеально підходить для централізації та опрацювання даних, а також стає у пригоді програмістам машинного навчання для роботи з часовими рядами та структурованими масивами даних [4].

6. TensorFlow

TensorFlow - це безкоштовна бібліотека з відкритим кодом, доступна для Python, JavaScript, C++ і Java. Ця універсальність дозволяє використовувати її у різних галузях з численними застосуваннями.

Початково розроблена командою Google Brain для внутрішніх потреб Google у дослідженнях та виробництві, перша версія була опублікована під ліцензією Apache 2.0 у 2015 році. У вересні 2019 року Google випустила оновлену версію під назвою TensorFlow 2.0.

Хоча TensorFlow може бути застосована для різних завдань, вона особливо ефективна у навчанні глибоких нейронних мереж.

Розробники, використовуючи TensorFlow, можуть створювати та навчати моделі машинного навчання не лише на комп'ютерах, але й на мобільних пристроях і серверах за допомогою TensorFlow Lite та TensorFlow Serving. Ці рішення пропонують ті ж самі можливості, але для мобільних платформ і високопродуктивних серверів [5].

7. Keras

Keras, відкрита бібліотека нейронних мереж, широко використовується для створення та оцінки моделей машинного та глибокого навчання.

Спочатку розроблена інженером Google для ONEIROS (Open-Ended Neuro Electronic Intelligent Robot Operating System), Keras швидко отримала підтримку в бібліотеках Theano та TensorFlow. Його можливість працювати з обома цими

фреймворками означала, що для навчання нейронних мереж не потрібно багато коду.

Ця бібліотека часто віддається перевагу через свою модульність, гнучкість та розширюваність, що робить її привабливим вибором для початківців [6].

Keras може легко інтегруватися з різними функціями, шарами, оптимізаторами та типами даних. Вона також пропонує один з найширших наборів типів даних і може працювати в різних оточеннях, включаючи обчислення як на процесорах, так і на відеокартах.

Отже, провівши аналіз кожної із найпопулярніших бібліотек можна зазначити, що найкраще для вирішення поставленої задачі підійдуть TensorFlow та Keras.

1.2. Методи навчання штучного інтелекту

В сучасному інформаційному суспільстві, питання пов'язані з обробкою великих обсягів даних набувають все більшого значення у різних сферах життєдіяльності. Ця обробка даних тісно переплітається з дослідженнями в галузі штучного інтелекту. Компанії-гіганти ІТ-індустрії, такі як Google, Microsoft, Apple, Amazon, активно розробляють і використовують алгоритми штучного інтелекту. Сфера штучного інтелекту включає багато напрямків, серед яких варто відзначити машинне навчання, нейромережі та глибоке навчання.

Одним із найбільш перспективних і складних напрямків є машинне навчання. Воно стало важливою складовою науки про дані, яка швидко розвивається. Машинне навчання допомагає розробляти алгоритми, які, за допомогою статистичних методів, навчаються класифікувати дані, робити прогнози та виділяти ключові моменти під час проєктів зі збору та аналізу даних.

Машинне навчання (Machine Learning - ML) – це значний підрозділ штучного інтелекту, який досліджує методи побудови алгоритмів, здатних самостійно навчатися. Сутність машинного навчання полягає у тому, що система аналізує велику кількість прикладів, виявляє закономірності та застосовує їх для

передбачення вихідних характеристик для нових вхідних даних. Основна ціль машинного навчання – частково або повністю автоматизувати вирішення складних аналітичних завдань [7]. Інструментальним набором для машинного навчання є алгоритми, які реалізовані певною мовою програмування та адаптовані до певного середовища та апаратного забезпечення. До популярних мов програмування для машинного навчання відносяться: C++, Octave, Python, Java, Ruby, R, Matlab, Scala.

Методи машинного навчання можна поділити за способом навчання на наступні категорії:

1. Навчання з учителем (Supervised Learning) – представляє собою одну з основних стратегій в галузі машинного навчання. В цьому підході ми надаємо моделі позначені навчальні набори даних, іншими словами - датасети, під наглядом експерта. Ці датасети містять інформацію про вхідні дані та відповідні бажані результати. Важливо відзначити, що значення вхідних та вихідних даних вже відомі.

Суть методу полягає в тому, що механізм штучного інтелекту намагається вивчити функцію відображення між цими вхідними та вихідними даними. Наприклад, якщо ми позначимо вихідні дані як Y , а вхідні дані як X , алгоритми машинного навчання намагаються знайти таку функцію відтворення f , що забезпечує відповідність $Y = f(X)$. Таким чином, модель намагається зрозуміти зв'язок між вхідними та вихідними даними на основі позначених прикладів з навчального набору.

Якщо розглядати це питання докладніше, то можна сказати, що у методі "навчання з учителем" контроль над процесом навчання моделі виконується спеціалістом з даних. Алгоритми намагаються побудувати функцію так, щоб передбачені значення (прогнози) були максимально близькі до фактичних правильних значень. Практично це означає, що алгоритми намагаються відобразити функцію, яка правильно асоціює вхідні дані X з вихідними даними Y' [8].

Під час навчання моделі відбувається процес порівняння прогнозних значень Y' з фактичними значеннями Y . Якщо різниця між цими значеннями стає меншою за певний поріг, то помилки стають прийнятними. Це означає, що модель досягла деякого рівня точності, і навчання зупиняється. Важливо зауважити, що хоча помилки можуть залишатися, але вони стають незначними.

Метод "навчання з учителем" дозволяє моделям навчатися ефективно і швидко, тому його широко використовують для вирішення практичних завдань.

Це особливо стосується завдань регресії (прогнозування числових значень) та класифікації (визначення категорії або класу для даних). Прикладом класифікації може слугувати модель, яка аналізує геометричні фігури на прикладі нижче (рис. 1.1). Зображення форми є розміченими вхідними даними для моделі, а розмітка цих зображень як їх опис, цей процес називається анотацією, є результуючими даними. Покладаючись на ці вхідні і вихідні дані система штучного інтелекту навчається прогнозувати вид отриманого зображення у відповідності до реального опису фігур [9].



Рис. 1.1. Навчання з вчителем

2. Навчання з підкріпленням (Reinforcement Learning) – це метод навчання, який зазвичай використовується для тренування автономних машинних агентів у різних середовищах. Цей підхід використовує моделі накопичувальної винагороди, щоб навчити агентів приймати рішення в різних

ситуаціях. Алгоритми спрямовані на максимізацію нагороди та зменшення ризику, в результаті чого відбувається навчання.

Застосування навчання з підкріпленням поширене в промислових галузях, а також стає все популярнішим у сфері багатокористувацьких онлайн-ігор. Середовище, в якому агент навчається, може бути будь-яким: реальний фізичний світ, симуляція або навіть віртуальна комп'ютерна гра.

Приклади застосування включають навчання роботів грати в складних іграх, таких як Dota 2, просто підкидаючи їх в гру та дозволяючи вчитися взаємодіяти з навколишнім світом. Також цей метод використовується для навчання систем автопілотування, наприклад, автомобільного автопілота Tesla, який в симуляціях навчається уникати зіткнень з пішоходами [10].

Роботам дійсно корисні знання про оточуюче середовище, але їм не обов'язково володіти усім світовими знаннями, оскільки вони можуть створювати та моделювати внутрішні ситуації. Основне завдання таких машин - не розраховувати всі можливі варіанти дій, а замість цього мінімізувати втрати або максимізувати показники успішності. Цей підхід нагадує реальний спосіб розвитку людей - вони навчаються на своїх помилках та отримують поштовх від правильних вчинків (рис. 1.2.).

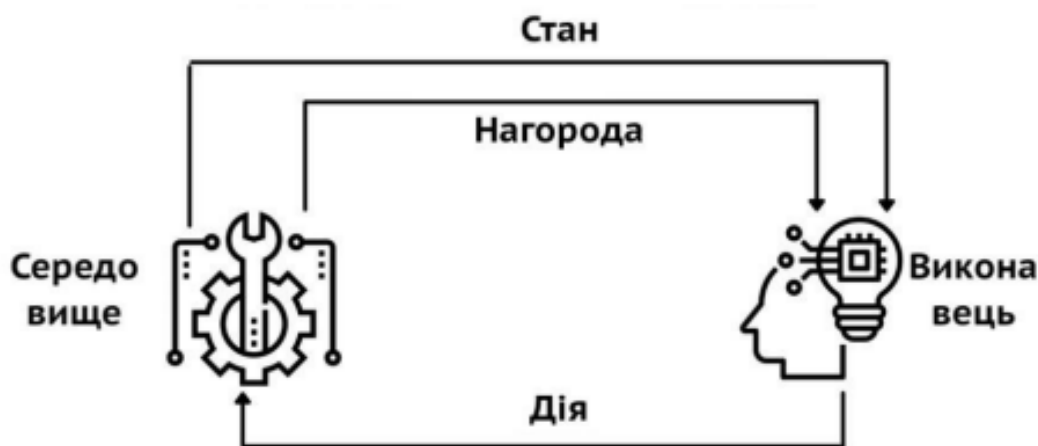


Рис. 1.2. Навчання з підкріпленням

3. Концепція навчання без вчителя (Unsupervised Learning) логічно випливає з самої назви: це підходи до навчання моделей, які оперують з неструктурованими наборами даних, не маючи прив'язаних "ідеальних"

результатів. У цій категорії машинного навчання ми надаємо моделі лише вхідні дані, не надаючи відповідних вихідних значень. В цьому контексті ми маємо вхідні дані, але нам невідомі результати та функція відображення.

У такому випадку система штучного інтелекту намагається знайти функцію, яка допоможе зрозуміти подібності між різними вхідними даними, шаблонами чи групами. Вона групує ці дані відповідно до рівня подібності, як показано на рис.1.3, створюючи псевдовихід для вивчення функції відповідності. Зазвичай такий підхід використовується для аналізу та підготовки даних, а не як головний алгоритм для знаходження конкретного рішення на основі даних [11].

На сьогоднішній день наявність добре анотованих даних у наборах є дуже цінною. Проте анотація є трудомісткою. Для цього використовують професійні сервіси або людські команди, або ж застосовують спеціальні алгоритми машинного навчання. Однак оптимальний підхід полягає у поєднанні автоматичної розмітки з використанням інших моделей штучного інтелекту, спільно з людськими ресурсами. Це допомагає прискорити процес анотації та покращити точність мережі.



Рис.1.3. Навчання без вчителя

Ці різні підходи в машинному навчанні дозволяють працювати з різними типами даних та вирішувати різноманітні завдання з прогнозування, класифікації та аналізу.

Для успішного використання алгоритмів машинного навчання важливо мати великий обсяг даних для навчання та тестування моделей. Також потрібно

проводити порівняльний аналіз ефективності різних побудованих моделей. Для цього використовуються вже наявні набори даних, які представлені у зручному форматі для використання та мають опубліковані результати роботи різних відомих алгоритмів.

Справді, хоча машинне навчання має великий потенціал для вирішення багатьох складних завдань, воно також постає перед численними викликами і проблемами. Однією з головних проблем є висока трудомісткість процесу побудови та налагодження систем машинного навчання. Це вимагає спеціалізованого знання з області машинного навчання та великої кількості робочих годин висококваліфікованих фахівців.

Додатково, впровадження машинного навчання може породжувати етичні питання. Поняття технологічної сингулярності, де штучний інтелект може перевищити інтелект людини, ставить питання щодо контролю над такими системами. Також може виникнути питання впливу на ринок праці, коли автоматизація завдань штучним інтелектом може призвести до змін в зайнятості.

Крім того, забезпечення конфіденційності та захисту особистих даних є актуальною проблемою у сфері машинного навчання, оскільки велика кількість даних збирається та використовується для навчання моделей.

З огляду на те, що машинне навчання продовжує швидко розвиватися, важливо досліджувати ці проблеми та шукати ефективні рішення для їх вирішення. Тільки з урахуванням цих аспектів можна досягти успішного та етичного впровадження машинного навчання в різні сфери життя.

1.3. Аналіз архітектур розпізнавання

Сучасні алгоритми комп'ютерного зору ґрунтуються на згорткових нейронних мережах, які принесли значне покращення продуктивності порівняно з традиційними методами обробки зображень.

Зазвичай згорткові нейронні мережі використовуються для завдань комп'ютерного зору, але їх також можна використовувати для аналізу тексту та

аудіо. Однією з перших архітектур CNN була AlexNet, яка здобула перемогу у конкурсі з візуального розпізнавання ImageNet у 2012 році.

Під час обробки зображень CNN використовує матриці значень для кожного базового кольору у зображенні, таких як червоний, зелений та синій. Ці значення оцінюються та агрегуються в тривимірні тензори для кольорових зображень, що представляють набори стеків карт функцій, пов'язаних з різними частинами зображення.

Ці тривимірні тензори формуються шляхом пропускання зображення через послідовність згорткових і пулінгових шарів. Згорткові шари виловлюють найбільш суттєві дані з різних частин зображення та згруповують їх у менші репрезентативні матриці. Цей процес повторюється кілька разів, в залежності від кількості згорткових шарів у конкретній архітектурі. Після витягнення характеристик з використанням згорткових шарів, отримані дані надсилаються на повністю зв'язаний (повнозв'язний) рівень, який генерує прогнози.

Ефективність та продуктивність згорткових нейронних мереж (CNN) обумовлені їхньою архітектурою [12]. Це включає в себе структуру шарів, конфігурацію елементів в кожному шарі та наявність специфічних компонентів. На протязі часу було розроблено багато різних архітектур CNN, серед яких особливо ефективними виокремлюються такі як:

1. AlexNet, яка була представлена у 2012 році, є однією з ключових архітектур, що взяла початок від попередньої LeNet і має велике значення для розвитку глибокого навчання. Вона впровадила важливі інновації, які допомогли вдосконалити продуктивність нейронних мереж у вирішенні завдань комп'ютерного зору.

AlexNet складається з п'яти згорткових шарів та трьох повнозв'язаних шарів. Однак її особливість полягає також у використанні подвійної конвеєрної структури, яка дозволила використовувати два графічних процесори (GPU) під час процесу навчання (рис.1.4). Це зробило навчання AlexNet значно швидшим та більш ефективним, що важливо при обробці великих обсягів даних, які зазвичай мають зображення [13].

Розробка AlexNet відкрила шлях до створення більш глибоких і складних архітектур, таких як VGG, ResNet, та інші, що стали основними компонентами успішних систем комп'ютерного зору.

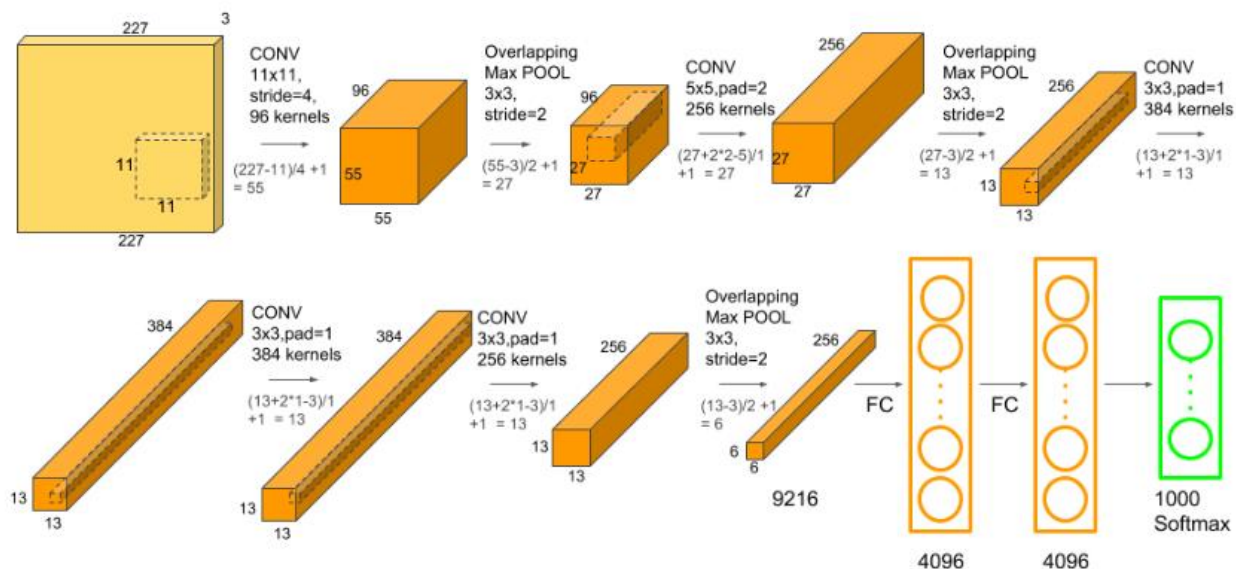


Рис. 1.4. Архітектура нейронної мережі AlexNet

2. GoogleNet (відома також як Inception V1), що була представлена у 2014 році. Вона відзначилася великою кількістю шарів і інноваційним підходом до згорткових мереж, що дозволило досягти високої продуктивності у вирішенні завдань комп'ютерного зору.

Однією з особливостей GoogleNet є використання початкових модулів, так званих "Inception modules". Ці модулі групуються в мережу і мають різні розмірності фільтрів, що дозволяє обробляти інформацію на різних рівнях деталізації та абстракції. Цей підхід допомагає зменшити кількість параметрів і обчислень, необхідних для навчання моделі та показано на рис.1.5.

Пакетна нормалізація і RMSprop відіграють важливу роль у покращенні процесу навчання. Пакетна нормалізація допомагає збалансувати розподіл значень у кожному батчі даних, що сприяє стабільному та швидкому навчання. RMSprop є методом адаптивної швидкості навчання, який регулює швидкість навчання для кожного параметра окремо, що допомагає покращити збіжність алгоритму навчання.

Загалом, GoogleNet стала важливим кроком у розвитку глибокого навчання і внесла значний внесок у вирішення завдань комп'ютерного зору та обробки зображень [14].

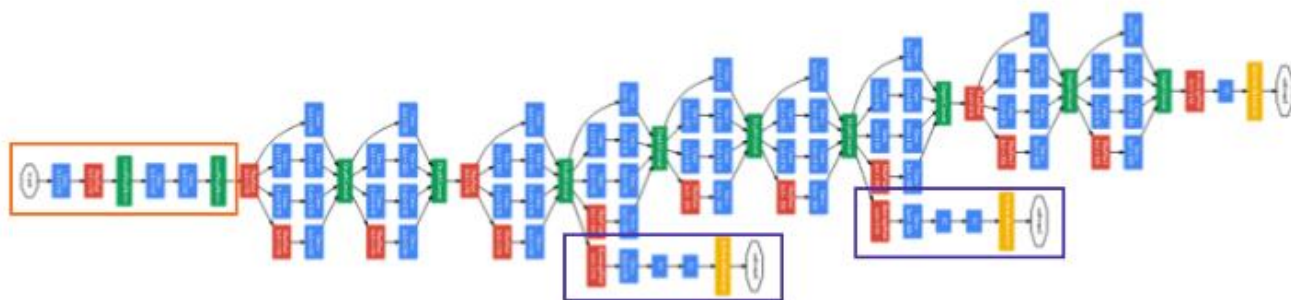


Рис.1.5. Архітектура нейронної мережі GoogleNet

3. VGGNet, яка була представлена у 2014 році на конкурсі візуального розпізнавання ImageNet. Ця архітектура (рис.1.6) славиться своєю глибиною і однорідною структурою, де використовуються лише згорткові шари розміром 3×3 і об'єднання розміром 2×2 .

Відмінністю VGGNet є те, що вона використовує низку менших згорткових фільтрів (3×3) замість великого фільтра (наприклад, 7×7), як це робилося в інших архітектурах. Такий підхід дозволяє збільшити глибину мережі, що допомагає моделі вчити більш абстрактні та складні риси зображень. Використання більше шарів дозволяє моделі отримати більше високорівневих функцій для розпізнавання об'єктів.

Зараз, завдяки розвитку глибокого навчання, відомо, що глибші мережі зазвичай мають більший потенціал для вивчення складних завдань, таких як розпізнавання об'єктів на зображеннях. VGGNet, хоч і не є найефективнішою архітектурою за кількістю параметрів, все ж є важливим внеском в розвиток глибокого навчання і комп'ютерного зору [15].

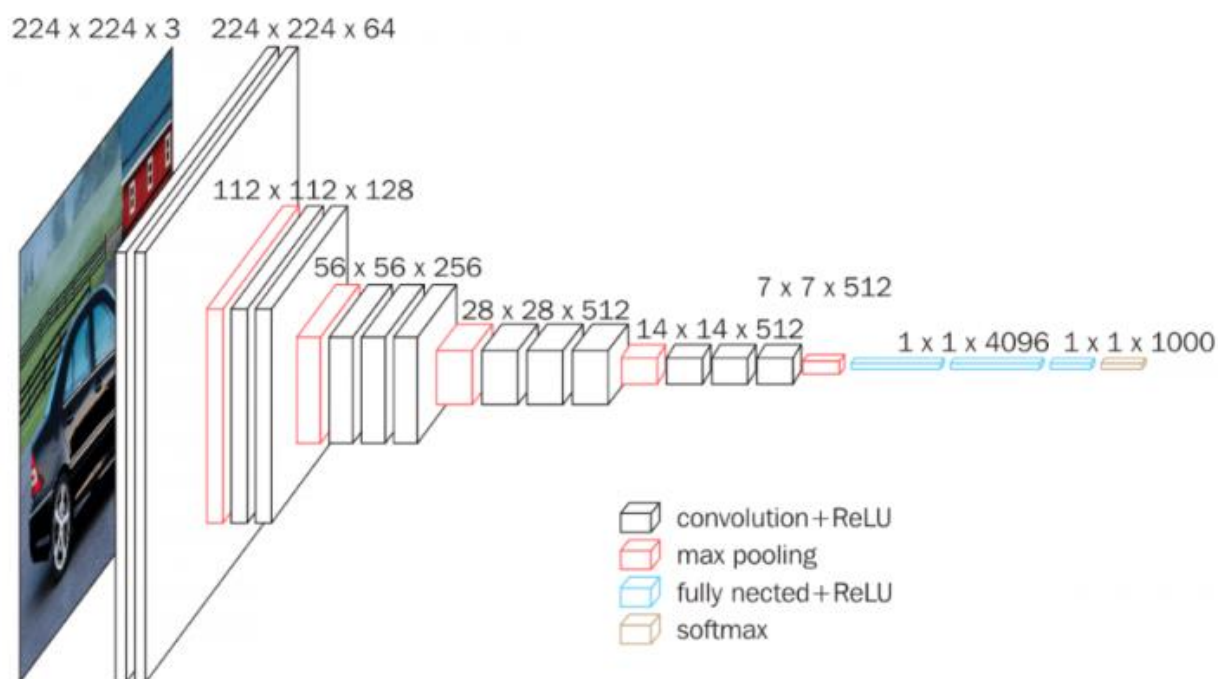


Рис.1.6. Архітектура нейронної мережі VGGNet

4. ResNet (Residual Network), яка була представлена у 2015 році. Однією з головних інновацій у ResNet є використання "пропуск з'єднань" (skip connections), також відомих як "residual connections". Ці з'єднання дозволяють пропустити вихід попереднього шару через поточний шар, що дозволяє зберегти та передати інформацію про зглиблюючі рівні мережі. Такий підхід допомагає уникнути проблеми зникаючих градієнтів і дозволяє збільшити глибину мережі.

Важливим аспектом ResNet (рис.1.7) є також використання пакетної нормалізації (batch normalization), що сприяє стабілізації процесу навчання. Пакетна нормалізація допомагає упоратися з проблемами, пов'язаними зі змінами розподілу вхідних даних на різних шарах мережі, і покращує збіжність процесу навчання [16].

ResNet визнана однією з найважливіших та найуспішніших архітектур у глибокому навчанні. Її використання дозволило побити рекорди в численних візуальних завданнях, включаючи конкурс ImageNet. Така архітектура дозволяє зберігати велику кількість шарів, що сприяє досягненню високої продуктивності і точності у різних завданнях комп'ютерного зору.

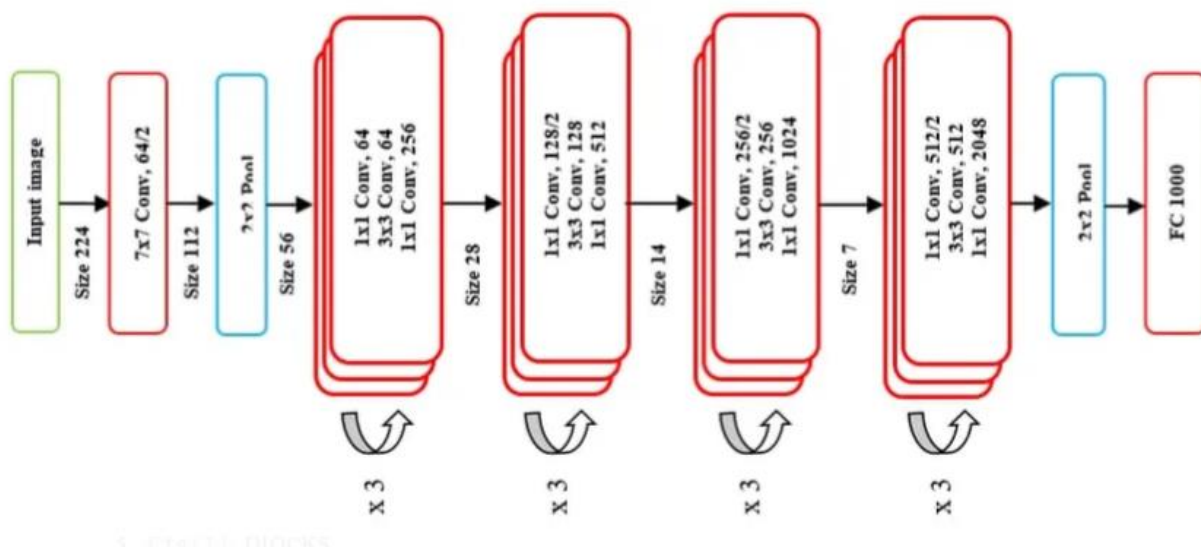


Рис.1.7. Архітектура нейронної мережі ResNet

5. YOLO (You Only Look Once) детектори об'єктів, що базуються на концепції високоефективного виявлення об'єктів на зображеннях і відео. Однією з ключових особливостей YOLO є здатність виконувати детекцію об'єктів в реальному часі, знижуючи час обробки кожного кадру і забезпечуючи швидкий відгук на вхідні дані.

Основна ідея YOLO полягає в тому, щоб одразу прогнозувати координати та класи об'єктів в кожній "клітинці" або "сітці" зображення. Сітка розділяє зображення на різні сегменти, і для кожного сегменту модель прогнозує координати та класи об'єктів, які можуть перебувати всередині нього. YOLO вирішує дві основні задачі: класифікацію об'єктів і прогнозування координат обмежувальних рамок, що оточують об'єкти. Архітектура мережі (рис.1.8) подібна до звичайних CNN, але на виході кожного шару модель видає класи ймовірностей і координати прямокутних рамок для об'єктів. Це дозволяє виявляти об'єкти різних класів та їхні положення на зображенні одночасно, а вся обробка виконується за один прохід.

Протягом років YOLO розвивалася, і вийшло декілька версій цієї моделі (починаючи з YOLO v1, яка була представлена у 2016 році, до більш нових

версій, таких як YOLOv8). Кожна наступна версія вносила покращення, які допомагали поліпшити якість детекції, точність та швидкодію моделі [17].

Короткий опис версій моделі YOLO:

- YOLOv2, випущена в 2016 році, покращила оригінальну модель, включивши пакетну нормалізацію, якірні вікна і кластери розмірів.

- YOLOv3, запущена в 2018 році, додатково покращила продуктивність моделі, використовуючи більш ефективну основну мережу, безліч якорів та пірамідальний просторовий пулінг [18].

- YOLOv4 була випущена в 2020 році, представивши такі інновації, як збільшення даних Mosaic, нову вільну від якірної голови детектування та нову функцію втрат.

- YOLOv5 далі покращила продуктивність моделі та додала нові функції, такі як оптимізація гіперпараметрів, інтегроване відстеження експериментів та автоматичний експорт у популярні формати.

- YOLOv6 була опублікована у відкритому доступі компанією Meituan у 2022 році і використовується в багатьох автономних роботах доставки компанії.

- YOLOv7 додала додаткові завдання, такі як оцінка пози за набором даних ключових точок COCO.

- YOLOv8 - остання версія YOLO від Ultralytics. Будучи передовою, сучасною моделлю, YOLOv8 заснована на успіху попередніх версій, впроваджуючи нові функції та вдосконалення для підвищення продуктивності, гнучкості та ефективності. YOLOv8 підтримує повний спектр завдань в галузі комп'ютерного зору, включаючи виявлення, сегментацію, оцінку пози, відстеження та класифікацію. Ця багатосторонність дозволяє користувачам використовувати можливості YOLOv8 у найрізноманітніших додатках та сферах діяльності.

Цей підхід робить YOLO дуже ефективним для використання в реальному часі, зокрема для задач виявлення об'єктів на відео або в потоках даних.

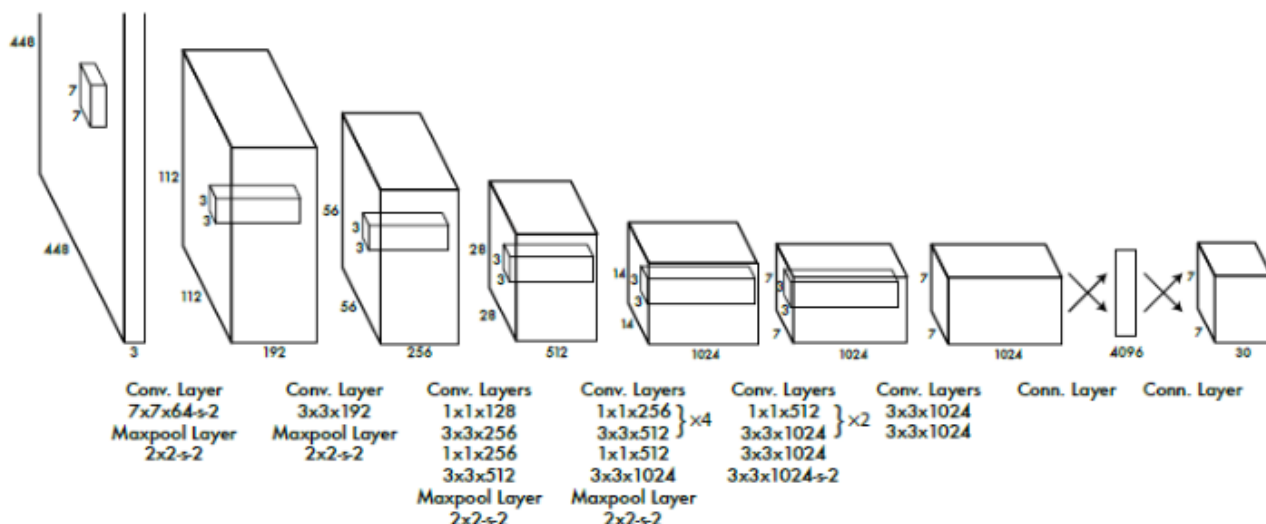


Рис.1.8. Архітектура нейронної мережі YOLO

1.4. Постановка завдання на розробку

Головним завданням є створення нейронної мережі для розпізнавання продукції. Виходячи з цього буде створена модель нейронної мережі для комп'ютерного зору та її розгортання у вигляді веб-додатку.

Для побудови нейромережі будуть реалізовані наступні кроки:

1. Створення датасету;
2. Поділ даних на навчальні та тестувальні;
3. Розробка та навчання нейронної мережі;
4. Розробка та розгортання веб-додатку
5. Тестування моделі для отримання найкращих результатів.
6. Порівняльний аналіз отриманих результатів
7. Тестування працездатності нейронної мережі у веб-додатку

Після отримання покрокової інструкції приступаємо до розробки згідно кожного пункту.

Висновки до першого розділу

В даному розділі було проаналізовано предметну область відповідно до теми роботи.

На першому етапі було проаналізовано існуючі бібліотеки мови програмування Python, які застосовуються для машинного навчання, а також для роботи із великими об'ємами даних.

Наступним кроком було оглянуто основні методи навчання штучного інтелекту. Це дало змогу краще ознайомитися з існуючими методами машинного навчання, зрозуміти основні переваги та недоліки кожного із методів навчання та охарактеризувати галузь до якої краще підходить той чи інший метод навчання.

Найбільшу увагу при аналізі предметної області було акцентовано саме на архітектурах розпізнавання. Даний огляд дав змогу дізнатися про основні типи архітектур, принцип їх дії та сфери застосування для кожної із них. Особливу увагу було приділено архітектурі YOLO, представлений у 2016 році. На даний час вона має 8 версій, що свідчить про її відмінну працездатність та інтерес розробників, відповідно до цього і стрімкий розвиток.

Завершальним етапом даного розділу було представлено завдання на розробку. У даному підрозділі покроково описано, що саме необхідно реалізувати, та у якій послідовності.

Даний розділ містить повний огляд предметної області та підґрунтя для подальшої розробки.

РОЗДІЛ 2. РОЗРОБКА СИСТЕМИ ДЕТЕКТУВАННЯ ЗА ДОПОМОГОЮ МОДЕЛІ YOLOV8N

2.1. Аналіз вхідних даних

Тренування нейронної мережі складається з багатьох частин, послідовність яких і складає пайплайн навчання. Найпершим кроком в цьому є описання проблематики завдання та повного формулювання цілі, які буде вирішувати отримана на виході нейронна мережа. Саме ціль робочої системи буде вирішувати, які методи та дані варто використовувати. В даній роботі висвітлюється робота над системою, що буде покращувати та полегшувати роботу у пошуку та детектуванні об'єктів на фото.

Цілі нейронної мережі:

- Аналіз отриманих даних у вигляді фото.
- Знаходження необхідних об'єктів на кожному екземплярі даних.
- Детектування об'єктів на тестових даних.



Рис.2.1. Робота нейронної мережі

Описуючи роботу майбутньої нейронки, можна виділити такий функціонал – знаходження на фото всіх пляшок торгівельної марки «Львівське

1715 »™ (рис. 2.1). Саме тому в навчальних даних для тренування нейронки в комп'ютерному зорі необхідно правильно підготовлювати інформацію.

Для реалізації даної системи комп'ютерного зору використаємо останню версію архітектури YOLO, а саме YOLOv8.

Ultralytics YOLOv8 — це не просто ще одна модель виявлення об'єктів; це універсальна структура, розроблена для охоплення всього життєвого циклу моделей машинного навчання — від прийому даних і навчання моделі до перевірки, розгортання та відстеження в реальному світі (рис.2.2). Кожен режим служить певній меті та створений для забезпечення гнучкості та ефективності, необхідних для різних завдань і випадків використання [19].

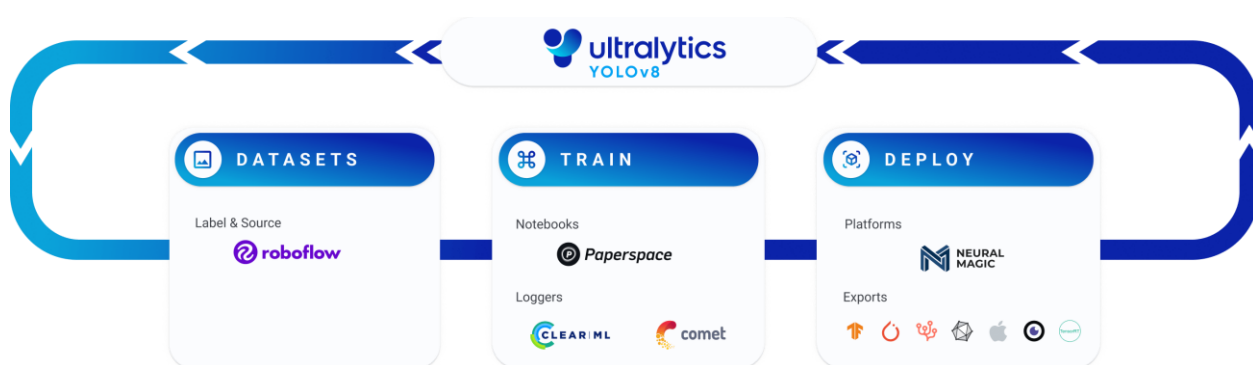


Рис.2.2. Ultralytics YOLOv8

Розуміння різних режимів, які підтримує Ultralytics YOLOv8, має вирішальне значення для отримання максимальної ефективності від ваших моделей:

- **Режим Train:** налаштуйте свою модель на основі спеціальних або попередньо завантажених наборів даних.
- **Режим Val:** контрольна точка після навчання для перевірки продуктивності моделі.
- **Режим Predict:** розкрийте потенціал прогнозування своєї моделі на реальних даних.
- **Режим Export:** підготуйте свою модель до розгортання в різних форматах.
- **Режим Track:** розширте свою модель виявлення об'єктів до програм відстеження в реальному часі.

– **Режим Benchmark:** аналізуйте швидкість і точність вашої моделі в різноманітних середовищах розгортання.

Для виконання поставленого завдання необхідно 2 режими – режим Train та Val.

Ось кілька переконливих причин використовувати режим Train YOLOv8:

– **Ефективність:** Використовуйте максимум можливостей вашого обладнання, будь то налаштування з одним GPU або розподіл навантаження на кілька GPU.

– **Універсальність:** Навчайте налаштовані набори даних, окрім вже доступних, таких як COCO, VOC та ImageNet.

– **Дружелюбний інтерфейс:** Простий, але потужний інтерфейс командного рядка (CLI) та Python для прямолінійного досвіду навчання.

– **Гнучкість гіперпараметрів:** Широкий спектр налаштовуваних гіперпараметрів для тонкого налаштування продуктивності моделі.

Також вказано основні переваги використання режиму Val:

– **Точність:** Отримайте точні метрики, такі як mAP50, mAP75 та mAP50-95, для всебічної оцінки вашої моделі.

– **Зручність:** Використовуйте вбудовані функції, які запам'ятовують налаштування навчання, спрощуючи процес валідації.

– **Гнучкість:** Перевіряйте вашу модель за допомогою тих самих або різних наборів даних та розмірів зображень.

– **Налаштування гіперпараметрів:** Використовуйте метрики перевірки для додаткового налаштування вашої моделі для кращої продуктивності [20].

2.2. Формування набору даних

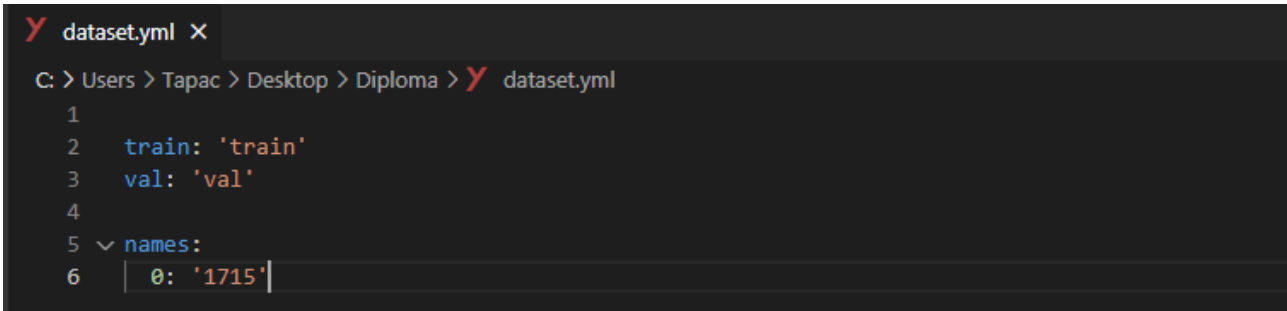
Другим не менш важливим кроком є аналіз проблематики знаходження даних для побудування датасету. Кожна система повинна отримати достатню кількість «якісної» та валідної інформації для створення певних закономірностей і паттернів для роботи у необхідній для роботи ситуації. Не виключенням є і розробка даної системи.

Основним завданням на даному етапі є збір достатньої кількості фото. Усі дані розділяються на навчальні дані та тестувальні.

Дане завдання можна розділити на два етапи:

1. Збір необхідної кількості даних.
2. Створення датасету

Маючи достатньо фото, потрібно скомпанувати їх в датасет. Для цього необхідно створити файл `dataset.yml` (рис.2.3), де розділи файли на тренувальні та валідаційні. Також, необхідно вказати назву класу.



```
Y dataset.yml X
C: > Users > Tapac > Desktop > Diploma > Y dataset.yml
1
2   train: 'train'
3   val: 'val'
4
5  names:
6    0: '1715'
```

Рис.2.3. Налаштування файлу `dataset.yml`

При приготуванні набору даних, необхідно підтримувати структуру збереження інформації для подальшого зручного використання, а також для правильного співвідношення метадати файлу і кожного зображення (рис.2.4).

Наступним важливим кроком у створенні навчальних даних, має бути розмітка підготовлених зображень у відповідності до цілі роботи згорткової нейронної мережі, а саме у відповідності до необхідних класів об'єктів.

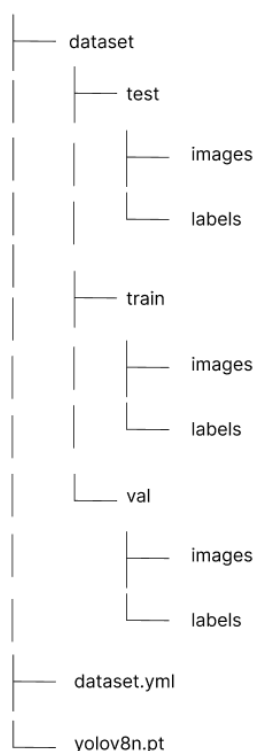


Рис. 2.4. Приклад правильної структури організації файлів у датасеті

Перед використанням софту для розмітки, варто розділити отриманий набір даних на тренувальну частину та валідаційну. При навчанні моделі варто розуміти принцип оцінювання «якісної» роботи нейронної мережі, а це відбувається за допомогою «прогання» тестової вибірки через отримані ваги на виході. Таким чином можна сказати, що результат роботи також залежить від правильного валідування CNN, тому що маючи максимально різноманітний і наближений до реального використання моделі тестовий набір даних можна наблизити до «ідеального» стану робочу систему.

Ще одним важливим фактором є унікальність частин `train` та `val`, тобто в другому наборі унікальні дані по відношенню до першого і немає перетину. Це не дає можливості нейронці на ранніх етапах при навчанні сформувати признаки, за якими буде відбуватись пошук об'єктів при тестуванні.

Розмітка зображення відображення в полі `detections->Detection` і містить в собі `id`, назву та обмежувальну рамку, нормалізовану по відношенню до розміру фото.

Для навчання моделі YOLOv8 необхідно перевести розмітку .txt файли для кожного зображення, де кожен рядок описує обмежувальну рамку.

```
0 0.207269 0.719731 0.063091 0.134457
0 0.169381 0.211348 0.065551 0.166279
0 0.912257 0.917426 0.068817 0.160042
0 0.230594 0.908906 0.063241 0.168838
0 0.233824 0.218329 0.069219 0.158840
0 0.288038 0.908013 0.062967 0.163006
0 0.299771 0.722024 0.060655 0.141142
0 0.857169 0.544709 0.068628 0.143160
0 0.864328 0.183573 0.070560 0.177643
0 0.411430 0.919003 0.068364 0.151770
0 0.193713 0.394328 0.050124 0.145641
0 0.853900 0.909313 0.067262 0.171542
0 0.787147 0.721064 0.066779 0.142447
0 0.712837 0.915996 0.065804 0.151007
0 0.590401 0.916637 0.064007 0.156620
0 0.791199 0.910557 0.067621 0.168993
0 0.371333 0.216319 0.062365 0.144334
0 0.807858 0.378045 0.064103 0.156416
0 0.786714 0.550608 0.072042 0.150634
0 0.796229 0.194059 0.068191 0.166151
0 0.533575 0.920247 0.061822 0.144519
0 0.725024 0.724099 0.064498 0.142483
0 0.667101 0.200264 0.064987 0.150798
```

Рис. 2.5. Приклад необхідного для тренування формату розмітки

На рис. 2.5. відображено необхідний формат для тренування. Першою цифрою виступає номер класу, за технічним завданням він один, тому там буде тільки 0. Далі йде чотири цифри, що вказують на розташування обмежувальної рамки відносно зображення, тобто перші дві цифри – це координати центру рамки, а два наступних – це ширина та висота рамки (рис. 2.6.). Координати нормалізовані та приведені до формату від 0 до 1.

Для формування необхідного формату було написано скрипт по обробці вхідних даних, що на виході дає необхідні дані для тренування. Важливо отримати два значення:

- labels, де зберігається кожен txt-файл з обмежувальними рамками;
- images, де лежать створенні посилання на оригінальні фото з відповідними іменами до txt файлів.

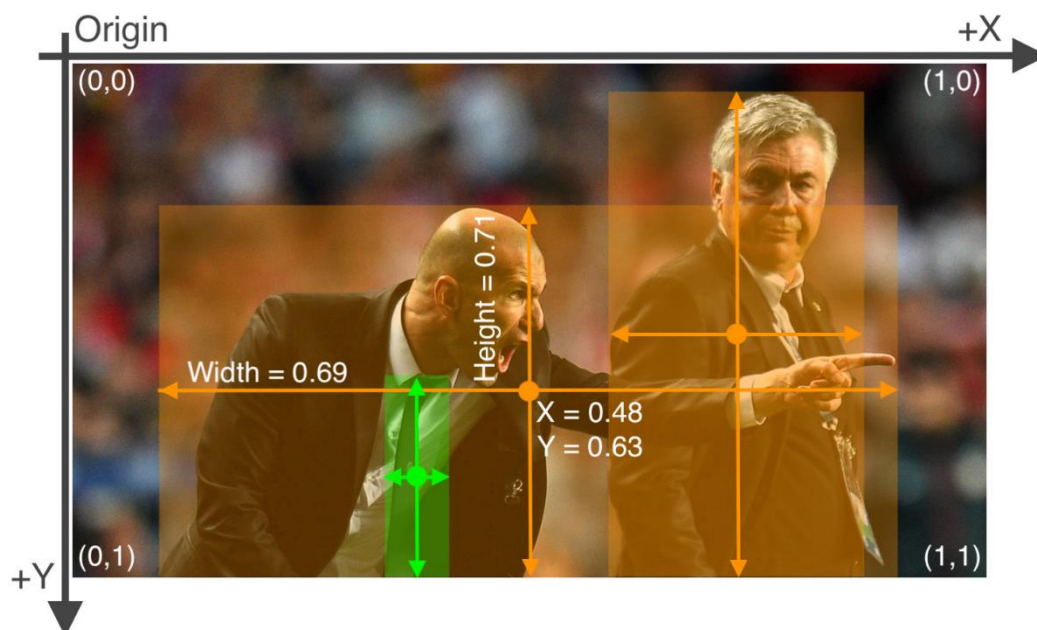


Рис. 2.6. Приклад опису обмежувальних рамок

2.3. Розробка та навчання моделі

Наступними кроками буде ітераційне тренування мережі з використанням створеного кастомного навчального набору даних. Процес навчання буде складатись з поступового навчання та валідації кожного кроку. Необхідно розбити тренувальний набір даних на частини.

Для отримання поступового результату, було отримано тренувальні дані та валідаційні. В абсолютних числах – це train в розмірі 125 зображень та val – 3 фото. При запуску тренування моделі ми отримаємо такий вивід до терміналу. На рис. 2.7 можна побачити ініціалізацію ваг архітектури YOLOv8n, де передається структура та встановлюються відповідні гіперпараметри.

```

Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8n.pt to 'yolov8n.pt'...
100%|██████████| 6.23M/6.23M [00:05<00:00, 1.20MB/s]
Ultralytics YOLOv8.0.227 Python-3.10.4 torch-2.1.2+cpu CPU (Intel Core(TM) i5-9400F 2.90GHz)
engine\trainer: task=detect, mode=train, model=yolov8n.pt, data=dataset.yaml, epochs=10, patience=50, batch=16, imgsz=640, save=True, save_period=-1, ca
WARNING:tensorflow:From c:\Users\Tapac\AppData\Local\Programs\Python\Python310\lib\site-packages\keras/src\losses.py:2976: The name tf.losses.sparse_so
Overriding model.yaml nc=80 with nc=1

      from  n  params module                                arguments
0         -1  1     464 ultralytics.nn.modules.conv.Conv    [3, 16, 3, 2]
1         -1  1    4672 ultralytics.nn.modules.conv.Conv    [16, 32, 3, 2]
2         -1  1    7360 ultralytics.nn.modules.block.C2f    [32, 32, 1, True]
3         -1  1   18560 ultralytics.nn.modules.conv.Conv    [32, 64, 3, 2]
4         -1  2   49664 ultralytics.nn.modules.block.C2f    [64, 64, 2, True]
5         -1  1   73984 ultralytics.nn.modules.conv.Conv    [64, 128, 3, 2]
6         -1  2  197632 ultralytics.nn.modules.block.C2f    [128, 128, 2, True]
7         -1  1  295424 ultralytics.nn.modules.conv.Conv    [128, 256, 3, 2]
8         -1  1  460288 ultralytics.nn.modules.block.C2f    [256, 256, 1, True]
9         -1  1  164608 ultralytics.nn.modules.block.SPPF    [256, 256, 5]
10        -1  1         0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
11        [-1, 6] 1         0 ultralytics.nn.modules.conv.Concat    [1]
12        -1  1  148224 ultralytics.nn.modules.block.C2f    [384, 128, 1]
13        -1  1         0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
14        [-1, 4] 1         0 ultralytics.nn.modules.conv.Concat    [1]
15        -1  1   37248 ultralytics.nn.modules.block.C2f    [192, 64, 1]
16        -1  1   36992 ultralytics.nn.modules.conv.Conv    [64, 64, 3, 2]
17        [-1, 12] 1         0 ultralytics.nn.modules.conv.Concat    [1]
...

```

Рис. 2.7. Ініціалізація мережі при тренуванні

Після формування архітектури, встановлюються шляхи до навчальних та валідаційних даних (рис.2.8).

```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
train: Scanning C:\Users\Tapac\Desktop\pivo\datasets\train.cache... 125 images, 2 backgrounds, 0 corrupt: 100%|██████████| 127/127 [00:00<?, ?it/s]
val: Scanning C:\Users\Tapac\Desktop\pivo\datasets\val.cache... 3 images, 1 backgrounds, 0 corrupt: 100%|██████████| 4/4 [00:00<?, ?it/s]
Plotting labels to runs\detect\yolov8n_1715\labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatically...
optimizer: AdamW(lr=0.002, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0.0)
Image sizes 640 train, 640 val
Using 0 dataloader workers
Logging results to runs\detect\yolov8n_1715

```

Рис.2.8. Відображення встановлення шляхів до навчальних даних

Налаштувавши середовище, програма запускає тренування моделі на 10 епох з розміром пакетів даних в 125 фото та за розміром 640 x 640 пікселів.

Наступний рисунок 2.9. відображає процес навчання YOLOv8n на створеному датасеті. На кожній ітерації можна побачити результати проміжних етапів тренування моделі. Зазвичай вони не дуже інформативні і направлені на надавання інформації про конкретний пакет навчальних даних, що не дає підстав робити заключення про вдалий тест чи ні.

Starting training for 10 epochs...
Closing dataloader mosaic

Epoch	GPU_mem	box_loss	cls_loss	df1_loss	Instances	Size	
1/10	0G	1.705	3.112	1.667	460	640: 100%	8/8 [00:52<00:00, 6.54s/it]
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%
	all	4	122	0.0992	0.975	0.232	0.165

Epoch	GPU_mem	box_loss	cls_loss	df1_loss	Instances	Size	
2/10	0G	0.9856	1.742	1.064	465	640: 100%	8/8 [00:32<00:00, 4.08s/it]
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%
	all	4	122	0.101	0.992	0.824	0.596

Epoch	GPU_mem	box_loss	cls_loss	df1_loss	Instances	Size	
3/10	0G	0.921	1.005	1.023	396	640: 100%	8/8 [00:33<00:00, 4.15s/it]
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%
	all	4	122	0.938	0.624	0.926	0.694

Epoch	GPU_mem	box_loss	cls_loss	df1_loss	Instances	Size	
4/10	0G	0.837	0.8354	0.999	480	640: 100%	8/8 [00:33<00:00, 4.22s/it]
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%
	all	4	122	0.913	0.601	0.924	0.704

Epoch	GPU_mem	box_loss	cls_loss	df1_loss	Instances	Size	
5/10	0G	0.7948	0.7357	0.9982	468	640: 100%	8/8 [00:33<00:00, 4.15s/it]
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%
	all	4	122	0.957	0.546	0.929	0.71

Epoch	GPU_mem	box_loss	cls_loss	df1_loss	Instances	Size	
6/10	0G	0.7957	0.7531	0.996	395	640: 100%	8/8 [00:33<00:00, 4.17s/it]
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%
	all	4	122	0.957	0.911	0.972	0.745

Рис. 2.9. Відображення тренування та проходження навчальних епох

По завершенню навчання відображається загальна інформація про тренування, а саме вказується кількість пройдених епох, час тренування та вказуються шляхи до вихідних ваг моделі (рис. 2.10). Є два вихідні файли best.pt та last.pt. Алгоритм адаптований для зберігання найкращої моделі в навчанні та останньої.

10 epochs completed in 0.102 hours.
Optimizer stripped from runs\detect\yolov8n_1715\weights\last.pt, 6.2MB
Optimizer stripped from runs\detect\yolov8n_1715\weights\best.pt, 6.2MB

Validating runs\detect\yolov8n_1715\weights\best.pt...

Ultralytics YOLOv8.0.227 Python-3.10.4 torch-2.1.2+cpu CPU (Intel Core(TM) i5-9400F 2.90GHz)

Model summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs

Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%	
all	4	122	0.953	0.988	0.984	0.807	1/1 [00:00<00:00, 1.58it/s]

Speed: 20.5ms preprocess, 50.3ms inference, 0.0ms loss, 4.1ms postprocess per image
Results saved to runs\detect\yolov8n_1715

Рис. 2.10. Результат навчання моделі

В результаті маємо наступну матрицю плутанини, яка показана на рисунку 2.11.

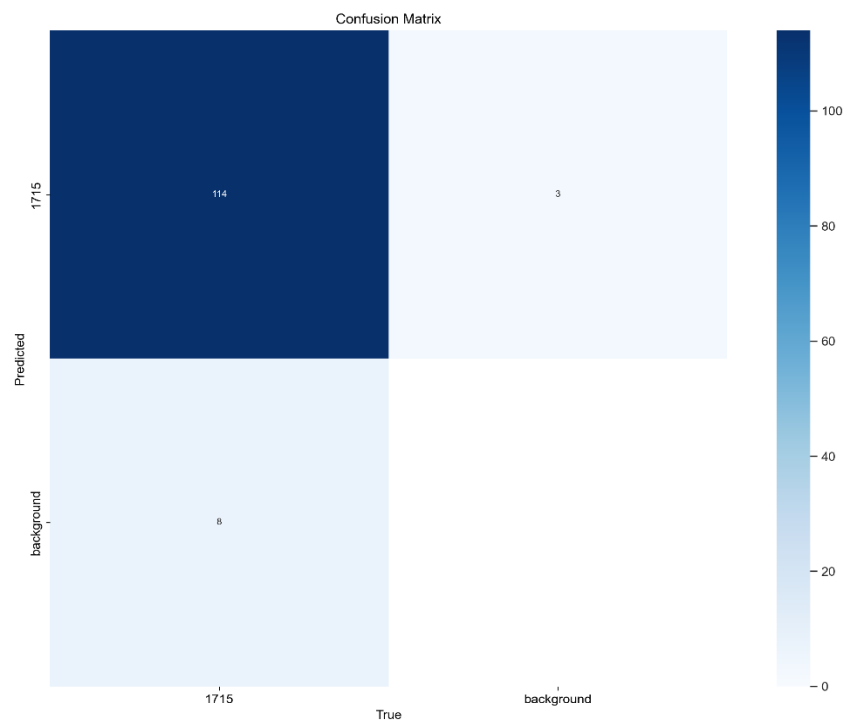


Рис.2.11. Матриця плутанини

На рисунку 2.12. зображена нормалізована матриця плутанини, які результат не може бути більше 1 З нормалізованої матриці плутанини ми можемо бачити, що результат навчання з 10 епохами становить 0.93, що складає 93% точності системи.

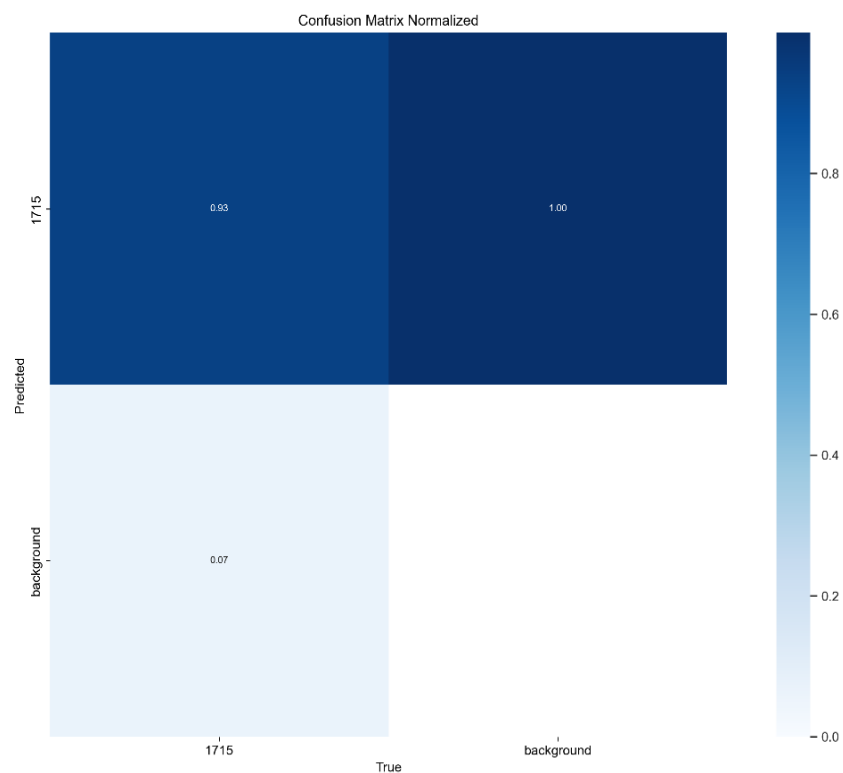


Рис.2.12. Нормалізована матриця плутанини

З кривої передбачення (рис.2.13) ми можемо бачити, що результат навчання з 10 епохами становить 0.607.

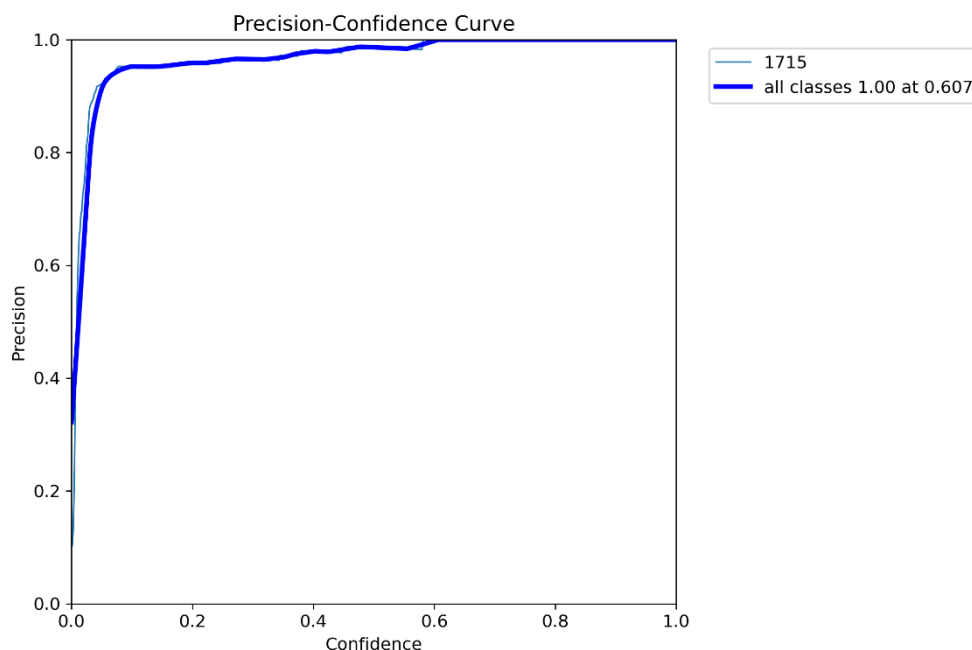


Рис.2.13. Крива передбачення

З результату порівняння можна зробити заключення, що навчання моделі на створених даних дає гарний приріст до робото-спроможності моделі. Проте також варто переглянути метрики на графіках для всіх тренувальних епох, щоб переконатись в правильності побудованого заключення (рис. 2.14).

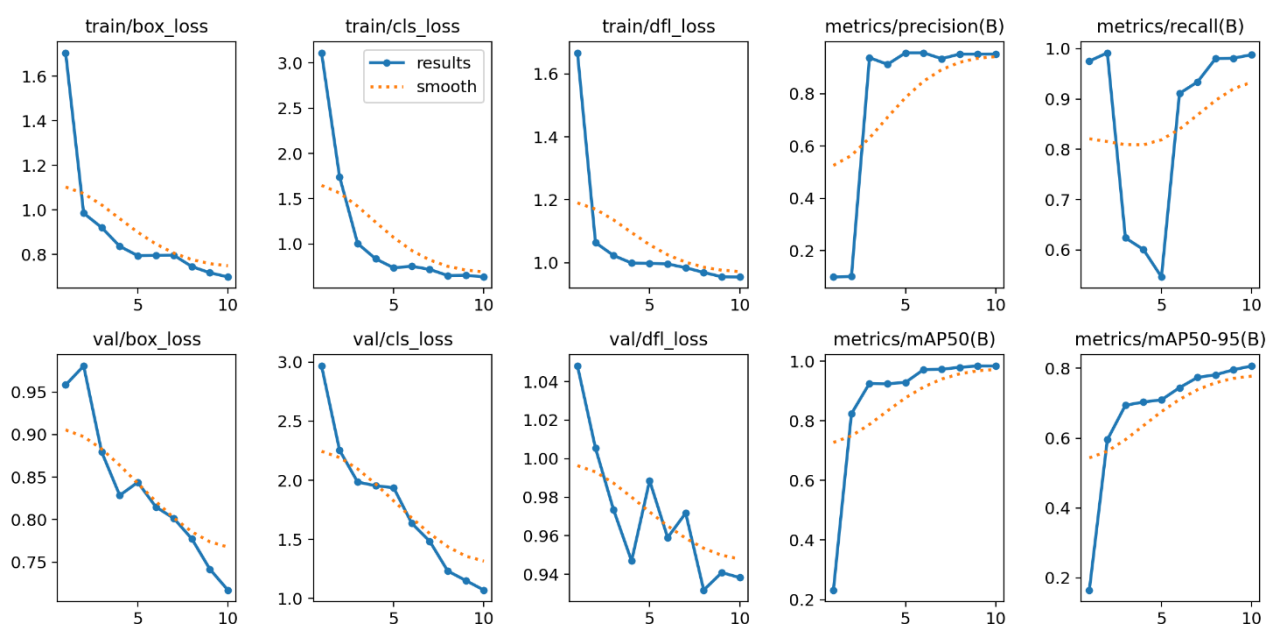


Рис.2.14. Графіки метрик навчання моделі

Маючи результати головних метрик за тренуванням, можна підтвердити, що тренування приносить достатньо хороший результат. Тому є підстави стверджувати, що набір даних сформований достатньо добре і дає можливість отримати потрібний результат на виході. Повний лістинг програми наведено у додатку А.

Всі результати після кожної епохи було занесе у файл results.csv (рис.2.15)

epoch,	train/box_loss,	train/cls_loss,	train/df_l_loss,	metrics/precision(B),	metrics/recall(B),	metrics/mAP50(B),	metrics/mAP50-95(B),	val/box_loss,	val/cls_loss,			
1,	1.7053,	3.1122,	1.6667,	0.09917,	0.97541,	0.23216,	0.16503,	0.95843,	2.9669,	1.0481,	0.00014,	0.00014,
2,	0.98555,	1.7418,	1.0638,	0.10083,	0.9918,	0.82408,	0.59572,	0.98079,	2.2533,	1.0054,	0.0002703,	0.00027
3,	0.92104,	1.0046,	1.023,	0.93835,	0.62385,	0.92565,	0.69423,	0.8795,	1.9852,	0.97348,	0.00036892,	0.00036
4,	0.83699,	0.83544,	0.99898,	0.9128,	0.60064,	0.92429,	0.70355,	0.82854,	1.9536,	0.94692,	0.00043586,	0.00043
5,	0.79479,	0.73574,	0.99817,	0.95691,	0.54616,	0.92897,	0.70977,	0.8439,	1.9366,	0.98862,	0.00047112,	0.00047
6,	0.79574,	0.75307,	0.996,	0.95695,	0.91109,	0.972,	0.74461,	0.81505,	1.6365,	0.95905,	0.0004747,	0.00047
7,	0.79676,	0.7181,	0.98417,	0.9344,	0.93398,	0.97297,	0.7741,	0.80148,	1.4836,	0.97178,	0.0004466,	0.00044
8,	0.74581,	0.65061,	0.969,	0.95223,	0.98033,	0.97958,	0.78152,	0.77734,	1.2322,	0.93181,	0.00038682,	0.00038
9,	0.71813,	0.65344,	0.95531,	0.95227,	0.98118,	0.98408,	0.79601,	0.74168,	1.1519,	0.9409,	0.00029536,	0.00029
10,	0.69935,	0.635,	0.95482,	0.95258,	0.98802,	0.98368,	0.80681,	0.71708,	1.0709,	0.93829,	0.00017222,	0.00017

Рис.2.15. Результати навчання після 10 епох

2.4. Проектування та розробка веб-додатку

Архітектура веб-додатків — це концептуальна модель, яка розкриває взаємозв'язок та взаємодію компонентів веб-програми. Вона визначає, як складові частини системи взаємодіють та розподіляють функції для оптимальної продуктивності та простоти розширення.

Один з найпоширеніших підходів до архітектури веб-додатків - це модель клієнт-сервер. У цій моделі, веб-додаток складається з двох головних складових: клієнтської та серверної. Клієнт (зазвичай веб-браузер) відображає інтерфейс користувача та збирає введені дані, тоді як сервер обробляє клієнтські запити, виконує бізнес-логіку та постачає необхідні дані.

Веб-додатки складаються з різних компонентів, які співпрацюють, щоб забезпечити функціональність. Клієнтська частина (Front-end) відповідає за відображення інтерфейсу користувача, тоді як серверна частина (Back-end) обробляє та відповідає на запити користувача.

Приклад архітектури веб-додатку показаний на рисунку 2.16.

У веб-додатковій архітектурі використовуються різні шаблони, які допомагають організувати та структурувати компоненти програми. Деякі з найпопулярніших архітектурних шаблонів - це монолітна, клієнт-серверна, розподілена та мікросервісна архітектура [21].

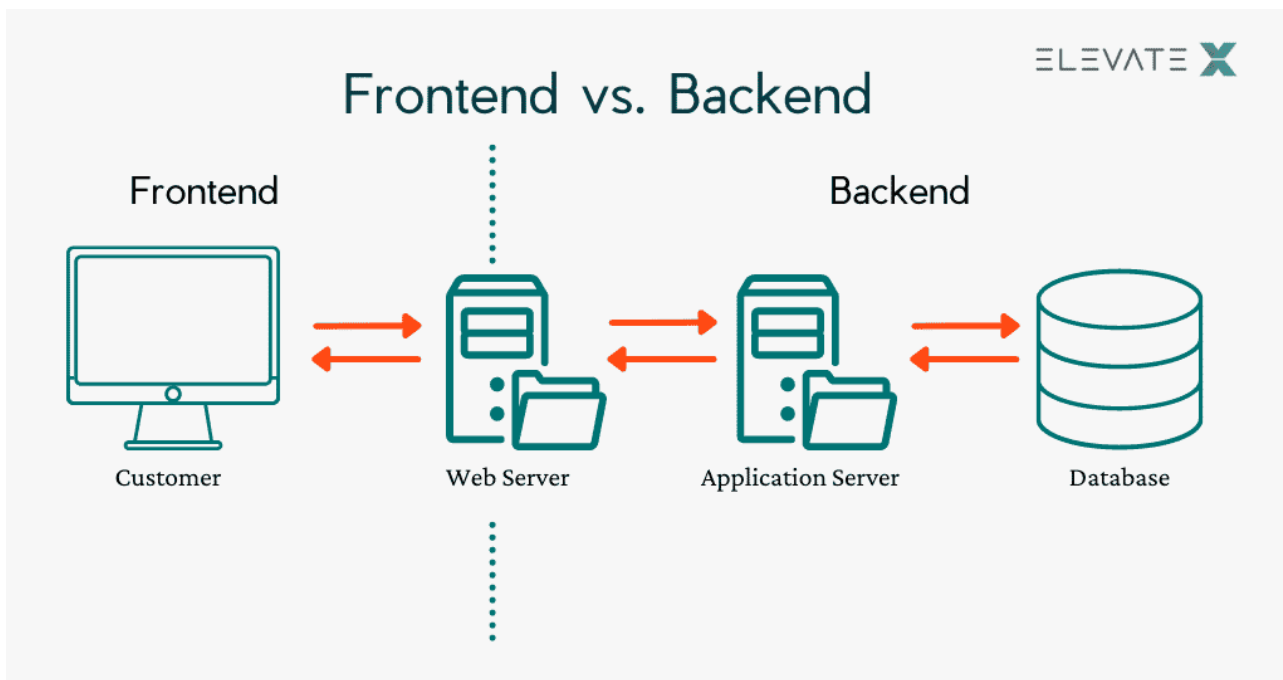


Рис.2.16. Архітектура веб-застосунка

Оскільки перше, з чим взаємодіє користувач це клієнтська частина, тому варто розпочати розробку саме із неї. Для того, щоб розробити клієнтську частина скористаємося JavaScript – бібліотекою для створення користувацьких інтерфейсів – React.js.

Фреймворк React.js – це фреймворк та бібліотека JavaScript з відкритим вихідним кодом, розроблені та підтримувані Facebook та Instagram. Він використовується для швидкого та ефективного створення інтерактивних інтерфейсів і веб-додатків із застосуванням значно меншої кількості коду, ніж при використанні звичайного JavaScript. У React ти розробляєш свої програми та проекти, створюючи повторно використовувані компоненти, які можна розглядати як незалежні блоки Lego. Ці компоненти являють собою окремі частини остаточного інтерфейсу, які у зібраному вигляді утворюють весь інтерфейс користувача.

Основна роль React (рис.2.17) у додатку — керування відображенням цієї програми так само, як літера V у шаблоні “модель-представлення-контролер” (MVC), забезпечуючи оптимальне та найефективніше виконання рендерингу. Замість того щоб розглядати весь інтерфейс користувача як єдине ціле, React.js пропонує розробникам розділити ці складні інтерфейси на окремі повторно використовувані компоненти, які утворюють будівельні блоки всього інтерфейсу користувача. При цьому платформа React JS поєднує швидкість і ефективність JavaScript з більш ефективним методом маніпулювання DOM для швидкого рендерингу веб-сторінок і створення високо динамічних і чуйних веб-додатків [22].



Рис.2.17. Логотип фреймворку React

Ознайомившись з основним засобом розробки фронтент частини перейдемо до розробки безпосередньо та структуризації файлів (рис.2.18).

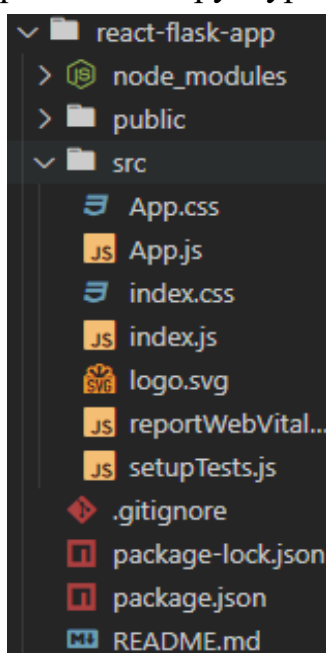
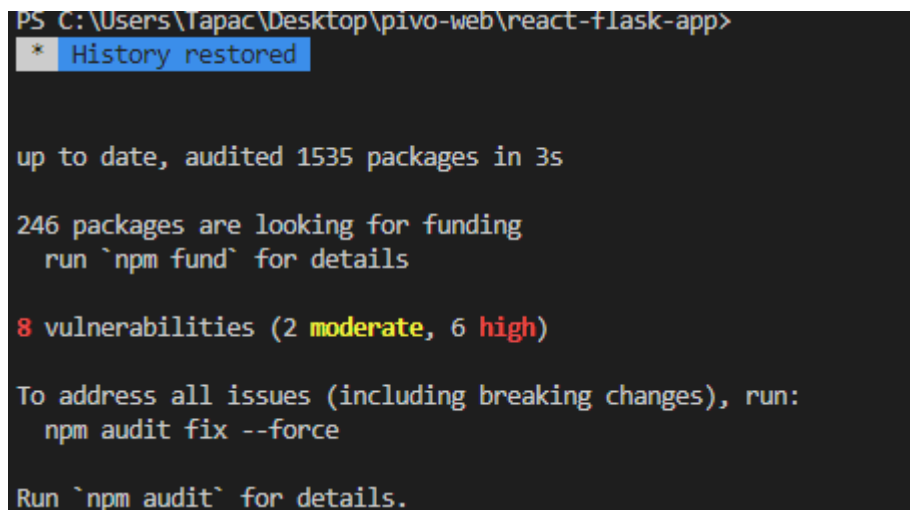


Рис.2.18. Структура клієнтської частини

Для того, щоб створити дану структуру необхідно скористатися командою `cd react-flask-app` у командному рядку.

Наступним кроком необхідно завантажити всі необхідні пакети. Це можливо реалізувати за допомогою команди `npm i` у командному рядку, як показано на рисунку 2.19.



```
PS C:\Users\Iapac\Desktop\pivo-web\react-flask-app>
* History restored

up to date, audited 1535 packages in 3s

246 packages are looking for funding
  run `npm fund` for details

8 vulnerabilities (2 moderate, 6 high)

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
```

Рис.2.19. Вигляд терміналу після команди `npm i`

Оскільки клієнтська частина для нас не грає важливої ролі, ми можемо пропустити вдосконалення зовнішнього вигляду за допомогою верстки. Основне завдання клієнтської частини це взаємодія із користувачем та демонстрування результату роботи створеної системи детектування. Варто лише продемонструвати файл `App.js` та його основні положення на рисунку 2.20.

```

import React, { useState } from 'react';
import axios from 'axios';

function App() {
  const [selectedFile, setSelectedFile] = useState(null);
  const [imageSrc, setImageSrc] = useState('');

  const handleFileChange = (event) => {
    setSelectedFile(event.target.files[0]);
  };

  const handleUpload = async () => {
    const formData = new FormData();
    formData.append('file', selectedFile);

    try {
      const response = await axios.post('http://localhost:5000/detect', formData, {
        responseType: 'arraybuffer' // change to arraybuffer to handle binary data
      });
      const base64 = btoa(
        new Uint8Array(response.data).reduce(
          (data, byte) => data + String.fromCharCode(byte),
          ''
        )
      );
      setImageSrc(`data:image/jpeg;base64,${base64}`);
    } catch (error) {
      console.error('Error uploading the file', error);
    }
  };

  return (
    <div className="App">
      <input type="file" onChange={handleFileChange} />
      <button onClick={handleUpload}>Upload and Detect</button>
      {imageSrc && <img src={imageSrc} alt="Detected" style={{ maxWidth: '1200px', maxHeight: '800px' }} />}
    </div>
  );
}

export default App;

```

Рис.2.20. Основні параметри файлу App.js

Функціональність клієнтської частини буде обмежуватися лише завантаженням необхідних даних та їх детектування, для чого виведено відповідне вікно та кнопку (Рис.2.21). Повний лістинг файлу App.js наведено у додатку Б.

```

      setImageSrc(`data:image/jpeg;base64,${base64}`);
    } catch (error) {
      console.error('Error uploading the file', error);
    }
  };

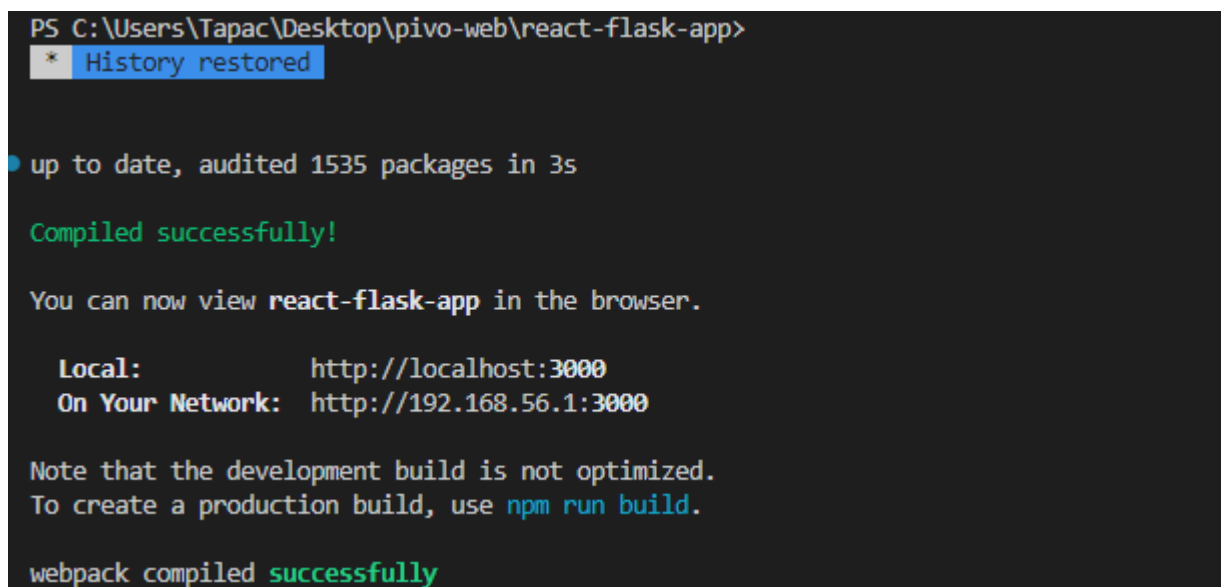
  return (
    <div className="App">
      <input type="file" onChange={handleFileChange} />
      <button onClick={handleUpload}>Upload and Detect</button>
      {imageSrc && <img src={imageSrc} alt="Detected" style={{ maxWidth: '1200px', maxHeight: '800px' }} />}
    </div>
  );
}

export default App;

```

Рис.2.21. Функціональність фронтенд частини

Для того, щоб запустити Frontend частину необхідно скористатися командою `npm start`. На рисунку 2.22 показано результат виконання даної команди у консолі.



```
PS C:\Users\Tapac\Desktop\pivo-web\react-flask-app>
* History restored

up to date, audited 1535 packages in 3s

Compiled successfully!

You can now view react-flask-app in the browser.

Local:      http://localhost:3000
On Your Network: http://192.168.56.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Рис.2.22. Вигляд терміналу після запуску Frontend частини

Зовнішній вигляд додатку у браузері показаний на рисунку 2.23.

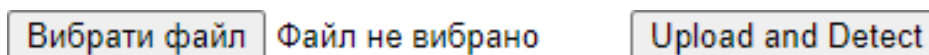


Рис.2.23. Зовнішній вигляд додатку у браузері

Наступним кроком необхідно створити Backend.

Backend - це термін, що використовується в розробці програмного забезпечення для опису компонентів системи, які відповідають за обробку даних та логіки за кулісами. У контексті веб-розробки, backend (задній кінець) відповідає за обробку запитів користувачів, виконання бізнес-логіки, зберігання та управління даними.

Backend включає в себе серверну частину веб-додатка. Це програмна складова, яка працює на сервері та відповідає за реалізацію функцій, які необхідні для взаємодії з клієнтами (наприклад, веб-браузерами). Backend може обробляти запити, виконувати розрахунки, взаємодіяти з базами даних, забезпечувати безпеку даних, генерувати вміст для відображення на фронтенді (клієнтській стороні).

Коли ви заходите на веб-сайт та виконуєте дії, такі як реєстрація, вхід у свій обліковий запис, заповнення форми або перегляд контенту, вся ця взаємодія із вами відбувається через взаємодію клієнта (фронтенд) із серверною частиною (бекенд). Фронтенд та бекенд співпрацюють, щоб надати вам повну функціональність та зручний користувацький досвід на веб-сайті чи додатку.

Для реалізації серверної частини буде використовувати мова програмування Python, а саме один із її фреймворків – Flask.

Flask - це легковаговий веб-фреймворк для Python, який пропонує базовий набір інструментів для розробки веб-додатків. Його можна використовувати як для створення простих лендінгів, так і для багатосторінкових веб-сайтів з великою кількістю додаткових модулів і сервісів [23].

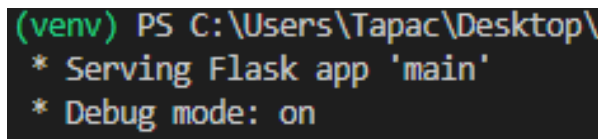
У Flask (рис.2.24) є численні переваги, що роблять його відмінним серед інших фреймворків:

- Простий синтаксис - він ґрунтується на мові програмування Python, що робить його дуже зрозумілим.
- Зручні шаблони - дозволяють швидко створювати прототипи веб-додатків.
- Широкий спектр інструментів для гнучкої настройки веб-сайтів під будь-які потреби.



Рис.2.24. Логотип фреймворку Flask

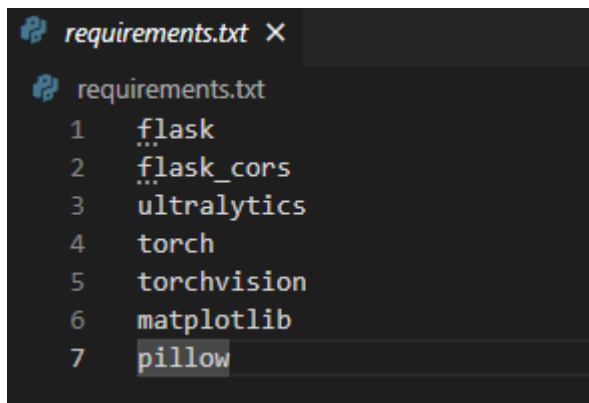
Для того, щоб запустити бекенд, перш за все необхідно створити середовище під проект. Для цього у командному рядку прописуємо команду `python -m venv venv`. Дана команда створить віртуальне середовище. Наступним кроком необхідно Активувати віртуальне середовище за допомогою команди `.\venv\Scripts\activate`. При правильній активації цієї команди робота буде продовжуватися уже безпосередньо в даному середовищі, як показано на рисунку 2.25.



```
(venv) PS C:\Users\Tapac\Desktop\  
* Serving Flask app 'main'  
* Debug mode: on
```

Рис.2.25. Активоване віртуальне середовище

Наступним кроком необхідно створити файл `requirements.txt`, де прописати всі основні залежності. Залежності прописати, як показано на рисунку 2.26. Та виконати ці залежності за допомогою команди `pip install -r .\requirements.txt` у командному рядку.



```
requirements.txt X  
requirements.txt  
1 flask  
2 flask_cors  
3 ultralytics  
4 torch  
5 torchvision  
6 matplotlib  
7 pillow
```

Рис.2.26. Залежності для віртуального середовища

Завершальним кроком запуску серверної частини є створення файлу `main.py` для завантаження створеної нейромережі у віртуальне середовище.

Перш за все, необхідно прописати всі бібліотеки, як показано на рисунку 2.27

```
main.py > detect
1  import json
2
3  from flask import Flask, request, send_file
4  from flask_cors import CORS
5  from ultralytics import YOLO
6  import os
7  import tempfile
8  import glob
```

Рис.2.27. Бібліотеки для файлу main.py

Також, підключаємо бібліотеку flask_cors. Цей пакет надає розширення Flask, яке за замовчуванням вмикає підтримку CORS на всіх маршрутах, для всіх джерел і методів. Це дозволяє параметризувати всі заголовки CORS на рівні кожного ресурсу. У комплект також входить декоратор, для тих, хто віддає перевагу такому підходу [24].

На наступному етапі підв'язуємо нашу нейронну модель YOLOv8n до нашого файлу, за допомогою команд, які показані на рисунку 2.28.

```
10  app = Flask(__name__)
11
12  CORS(app)
13
14  model_path = 'runs/detect/yolov8n_1715/weights/best.pt'
15  model = YOLO(model_path)
```

Рис.2.28. Виклик нейронної моделі до файлу

На рисунку 2.29. покажемо функцію, яка створена на наступному етапі розробки із методом POST, яка буде працювати безпосередньо із файлом, який завантажуватиметься.

```

18 @app.route('/detect', methods=['POST'])
19 def detect():
20     if 'file' not in request.files:
21         return "No file part", 400
22
23     file = request.files['file']
24     if file.filename == '':
25         return "No selected file", 400
26
27     if file:
28         # Використовуємо tempfile для створення тимчасового файла
29         with tempfile.NamedTemporaryFile(delete=False, suffix='.jpg') as tmp:
30             file.save(tmp.name)
31             # Виконуємо виявлення об'єктів
32             result = model.predict(tmp.name, save=True, conf=0.5)
33
34             # Видаляємо тимчасовий файл
35             os.remove(tmp.name)
36
37             image_path = os.path.join(os.path.curdir, result[0].save_dir, os.path.basename(result[0].path))
38
39             image = glob.glob(image_path)[0]
40
41             return send_file(image, mimetype='image/jpeg', as_attachment=True)

```

Рис.2.29. Функція для роботи із файлом

Повний лістинг файлу main.py буде наведено у додатку В.

Переглянути створену нейронну мережу та веб-додаток можливо за посиланням:

<https://git.ztu.edu.ua/ipz/magistry2022-2023/ipzm-22-2/predchuk-taras/diploma>

Висновки до другого розділу

В даному розділі було розроблено систему детектування за допомогою моделі YOLOv8n.

Створення моделі було розпочату із аналізу вхідних даних. Проаналізувавши вхідні дані було створено вимоги до нейронної мережі компю'терного зору. Необхідно провести аналіз отриманих даних у вигляді фото, наступним кроком було знаходження необхідних об'єктів на кожному екземплярі даних та завершальним кроком було детектування об'єктів на тестових даних.

Завершивши аналіз вхідних даних було розпочато етап формування датасету. На даному етапі розробки було створено власний датасет із унікальними даними та оформлено рамки для кожного фото у межах, необхідних для коректної роботи системи розпізнавання.

На наступному етапі роботи було створено нейронну мережу для детектування об'єктів на фото. Після вибору версії моделі YOLO, було описано основні гіперпараметри для створення та першого навчання системи. По завершенні навчання було отримано вихідні дані та графіки, які наведено у підрозділі 2.3.

Завершальним етапом даного розділу було створення веб-додатку для інтеграції у нього розробленої нейронної мережі детектування. У якості серверної частини було обрано фреймворк мови Python – Flask. Клієнтську частину було реалізовано за допомогою бібліотеки React.js.

При написанні даного розділу було дотриманось всім вимог згідно поставленого завдання.

РОЗДІЛ 3. ДОСЛІДЖЕННЯ ПРАЦЕЗДАТНОСТІ ДОДАТКУ ТА ЕФЕКТИВНОСТІ РІШЕННЯ ЗАДАЧІ РОЗПІЗНАВАННЯ

3.1. Оцінка та аналіз якості моделі

Оцінка та аналіз якості моделі є важливим етапом в розробці методу розпізнавання продуктів за допомогою нейронної мережі.

Першим кроком цього етапу є збір даних для тренування моделі та тестування. Важливо використовувати якісні дані, щоб забезпечити точність та ефективність моделі. Також можна використовувати методи підвищення якості даних, наприклад, методи аугментації даних, для покращення роботи моделі.

Далі необхідно обрати метрики для оцінки якості моделі, такі як точність, таємність, F-оцінка та інші. Залежно від конкретного застосування, можуть мати значення різні метрики. Наприклад, при застосуванні методу для визначення наявності продукту на полиці, важливо забезпечити високу точність, оскільки помилки можуть призвести до неправильної оцінки запасів та неефективного управління.

Крім того, можна використовувати додаткові методи для покращення якості моделі, такі як оптимізація параметрів нейронної мережі, підвищення її глибини, покращення алгоритму тренування та інші.

Після оцінки якості моделі, важливо провести аналіз результатів. Якщо якість моделі нижче, ніж очікувалося, можна звернутися до методів покращення, що були згадані вище, з метою подальшого покращення. Крім того, можна дослідити причини помилок та якщо є недоліки в даних, виправити їх.

Оскільки, під час навчання нейронна модель для розпізнавання показала результат у 93%, що є замалим показником для коректної роботи системи, тому наступним етапом роботи є доведення даної мережі до 99%. Для цього буде змінено основні гіперпараметри моделі і порівняно з іншими показниками під час минулих ітерацій навчання.

Розпочнемо із головного показника при навчання – кількість епох.

Епохи використовуються при тренуванні нейронних мереж як одиниці вимірювання проходження всього набору даних через мережу. Кожна епоха представляє собою одне повне проходження всієї навчальної вибірки через мережу нейронів під час процесу навчання.

Під час кожної епохи дані подаються до мережі, зворотні зв'язки обчислюються, ваги нейронів оновлюються і алгоритм навчання пристосовує модель до даних. Тренувальний процес може складатися з кількох епох, залежно від складності задачі, доступної кількості даних і характеристик самої моделі.

Використання епох у тренуванні нейронних мереж дозволяє контролювати і відстежувати процес навчання, визначати оптимальну кількість проходжень даних через мережу для досягнення найкращої продуктивності моделі.

Під час створення нейромережі параметр кількості епох був встановлений у кількості 10. При повторному навчанні встановимо цей показник на 13 (рис.3.1).

```
model = YOLO('yolov8n.pt')
results = model.train(data='dataset.yaml', imgsz=640, epochs=13, batch=16, name='yolov8n_1715')
```

1m 23.0s

Ultralytics YOLOv8.0.227 Python-3.10.4 torch-2.1.2+cpu CPU (Intel Core(TM) i5-9400F 2.90GHz)
engine\trainer: task=detect, mode=train, model=yolov8n.pt, data=dataset.yaml, epochs=13, patience=50, batch=16, imgsz=640
WARNING:tensorflow:From c:\Users\Tapac\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\losses.py:297

Overriding model.yaml nc=80 with nc=1

	from	n	params	module	arguments
0	-1	1	464	ultralytics.nn.modules.conv.Conv	[3, 16, 3, 2]
1	-1	1	4672	ultralytics.nn.modules.conv.Conv	[16, 32, 3, 2]
2	-1	1	7360	ultralytics.nn.modules.block.C2f	[32, 32, 1, True]
3	-1	1	18560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
4	-1	2	49664	ultralytics.nn.modules.block.C2f	[64, 64, 2, True]
5	-1	1	73984	ultralytics.nn.modules.conv.Conv	[64, 128, 3, 2]
6	-1	2	197632	ultralytics.nn.modules.block.C2f	[128, 128, 2, True]
7	-1	1	295424	ultralytics.nn.modules.conv.Conv	[128, 256, 3, 2]
8	-1	1	460288	ultralytics.nn.modules.block.C2f	[256, 256, 1, True]
9	-1	1	164608	ultralytics.nn.modules.block.SPPF	[256, 256, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
12	-1	1	148224	ultralytics.nn.modules.block.C2f	[384, 128, 1]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
15	-1	1	37248	ultralytics.nn.modules.block.C2f	[192, 64, 1]
16	-1	1	36992	ultralytics.nn.modules.conv.Conv	[64, 64, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]

Рис.3.1. Навчання нейронної мережі із 13-ма епохами

На рисунку 3.2. показана інформація, яку ми отримуємо після закінчення навчання моделі із 13 епохами.

```
13 epochs completed in 0.142 hours.
Optimizer stripped from runs\detect\yolov8n_17152\weights\last.pt, 6.2MB
Optimizer stripped from runs\detect\yolov8n_17152\weights\best.pt, 6.2MB

Validating runs\detect\yolov8n_17152\weights\best.pt...
Ultralytics YOLOv8.0.227 Python-3.10.4 torch-2.1.2+cpu CPU (Intel Core(TM) i5-9400F 2.90GHz)
Model summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs

```

Class	Images	Instances	Box(P)	R	mAP50	mAP50-95)
all	4	122	0.958	0.992	0.983	0.808

```
Speed: 17.1ms preprocess, 66.0ms inference, 0.0ms loss, 7.9ms postprocess per image
Results saved to runs\detect\yolov8n_17152
```

Рис.3.2. Результат навчання із 13 епохами

Наступним кроком необхідно проаналізувати графіки після навчання даної моделі. Це дасть змогу краще зрозуміти стан моделі та напрям куди рухатися далі.

Для початку, на рисунку 3.3. продемонструємо графік F1. Графік F1 confidence зазвичай демонструє залежність міри F1 від рівня впевненості моделі в її передбаченнях. Міра F1 є комбінованою метрикою точності й повноти, що використовується для оцінки якості моделі класифікації. Рівень впевненості моделі відображається на осі X, а значення F1 відображаються на осі Y.

Цей графік вказує на те, як змінюється ефективність моделі залежно від того, наскільки вона впевнена у своїх передбаченнях. Зазвичай, графік F1 confidence допомагає визначити поріг впевненості, який забезпечує оптимальний баланс між точністю та повнотою моделі. Наприклад, він може вказати на те, що при вищій впевненості модель працює точніше, але з меншою повнотою, тоді як при низькій впевненості точність може знизитися, але повнота зросте [25].

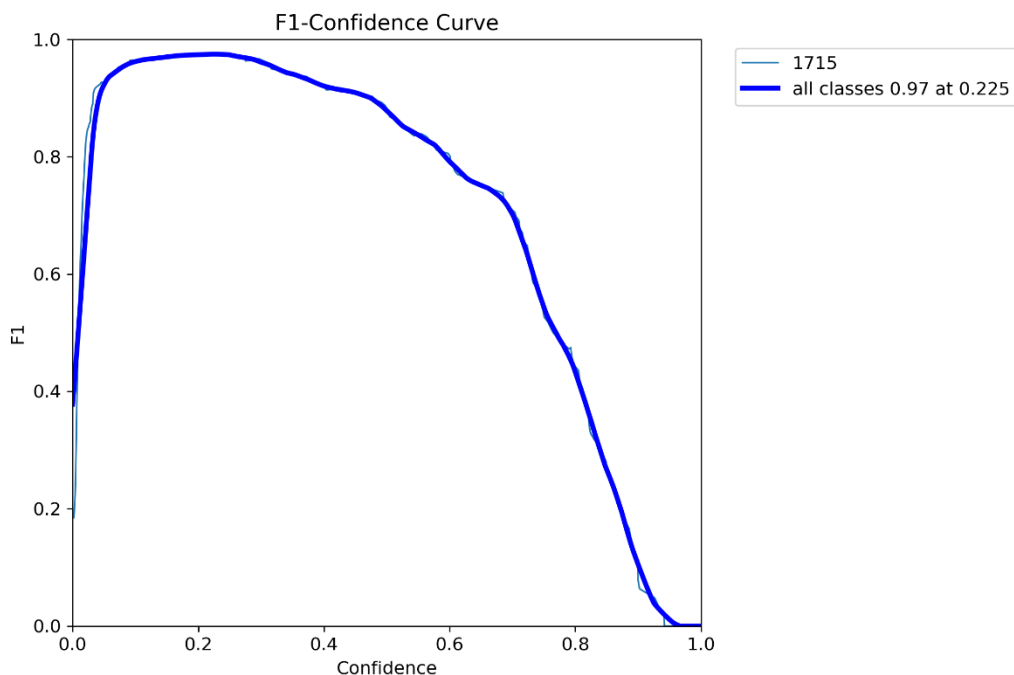


Рис.3.3 Крива F1

Виходячи із даного графіку можна побачити, що значення міри F1, яке вказує на якість моделі при певному рівні впевненості (у цьому випадку - 0.97). Високе значення F1 вказує на добру балансованість між точністю і повнотою моделі класифікації.

0.225 – це рівень впевненості, для якого отримано значення F1 - 0.97. Це означає, що при впевненості, рівній 0.225, модель показує високе значення F1, яке дорівнює 0.97.

Це може вказувати на те, що при даному порозі впевненості (0.225) модель виявляється дуже точною та збалансованою щодо своїх класифікаційних передбачень.

На рисунку 3.4. продемонструємо графік Precision-Confidence Curve.

Даний графік відображає залежність точності моделі від рівня впевненості цієї моделі в її класифікаційних передбаченнях. Він показує, як змінюється точність передбачень моделі для різних рівнів впевненості.

На цьому графіку по осі X зазвичай відображується рівень впевненості моделі, а по осі Y - точність передбачень, тобто відношення правильно

класифікованих прикладів до всіх прикладів, що модель віднесла до даного класу з певним рівнем впевненості.

Аналогічно до F1-кривої, Precision-Confidence Curve допомагає зрозуміти, як модель веде себе при різних рівнях впевненості. Наприклад, вона може показати, що при високій впевненості модель дає високу точність, але при зниженні впевненості точність може спадати. Це може бути корисним для встановлення оптимального порогу впевненості для моделі, щоб досягти бажаної точності при класифікації.

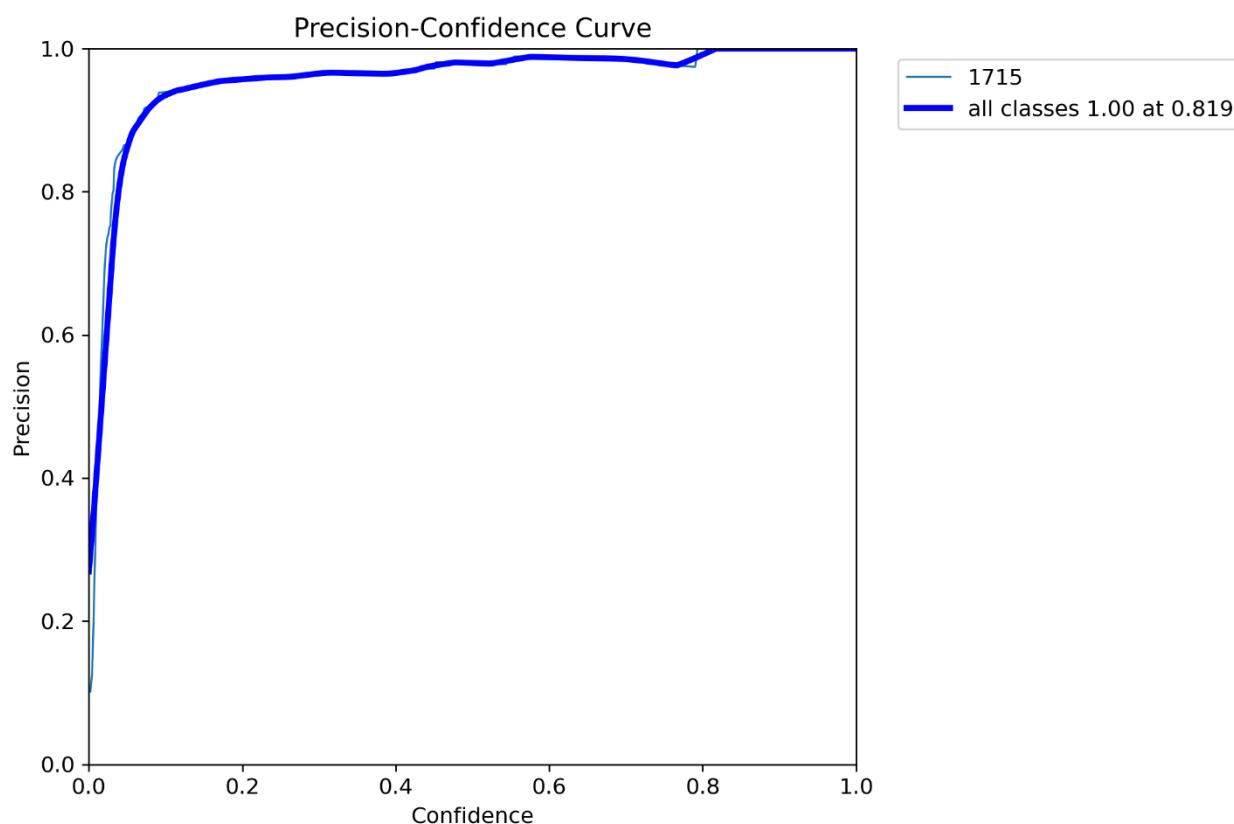


Рис.3.4. Precision-Confidence Curve для 13 епох навчання

1.00 – це значення точності моделі, яке вказує на якість класифікації при певному рівні впевненості (у цьому випадку - 1.00, або 100%). Максимальне значення точності вказує на те, що всі класифікаційні передбачення для цього рівня впевненості були вірними.

0.819 - це рівень впевненості моделі, для якого досягнуто значення точності 1.00. Це означає, що при впевненості, рівній 0.819, модель показує

максимальну точність - всі її класифікаційні передбачення для цього рівня впевненості є правильними.

Це вказує на те, що для цього конкретного рівня впевненості (0.819) модель показує ідеальну точність в своїх класифікаційних передбаченнях.

Наступним графіком покажемо матрицю плутанини (рис.3.5) та нормалізовану матрицю плутанини (рис.3.6).

Графік матриці плутанини (confusion matrix) візуалізує результати класифікації моделі, порівнюючи фактичні класи з передбаченими класами для кожного класу. Ця матриця дозволяє швидко оцінити продуктивність моделі, вказуючи на кількість правильно та неправильно класифікованих елементів.

Матриця має рядки та стовпці, які відповідають реальним класам і передбаченим класам відповідно. Типовий вигляд показана у таблиці 3.1.

Таблиця 3.1.

Фактичний Клас	Передбачено Клас 0	Передбачено Клас 1
Фактичний Клас 0	True Negative	False Positive
Фактичний Клас 1	False Negative	True Positive

У даній таблиці :

- True Positive (TP): Кількість правильно класифікованих елементів, які належать до позитивного класу.
- True Negative (TN): Кількість правильно класифікованих елементів, які належать до негативного класу.
- False Positive (FP): Кількість неправильно класифікованих елементів, які помічені моделлю як позитивний клас, але насправді вони належать до негативного класу (ложно позитивні).
- False Negative (FN): Кількість неправильно класифікованих елементів, які помічені моделлю як негативний клас, але насправді вони належать до позитивного класу (ложно негативні).

Цей графік є корисним інструментом для визначення того, в яких областях модель виявляється точною або недостатньою, допомагаючи виправити можливі помилки та покращити її продуктивність.

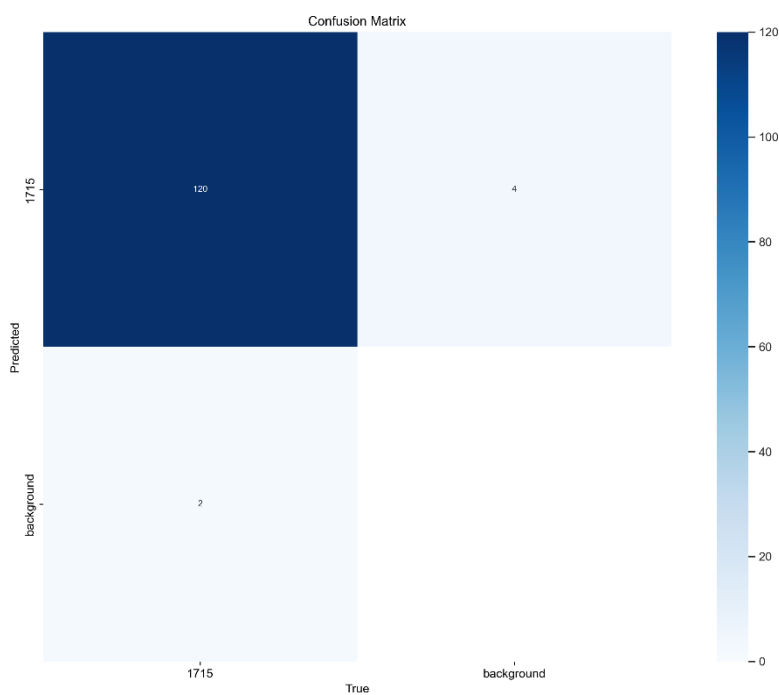


Рис.3.5. Матриця плутанини нейромережі із 13 епохами

Нормалізуємо матрицю плутанини. Нормалізація матриці плутанини - це процес перетворення числових значень у матриці плутанини таким чином, щоб вони відображали відносні частки помилок класифікації, а не абсолютні кількості помилок.

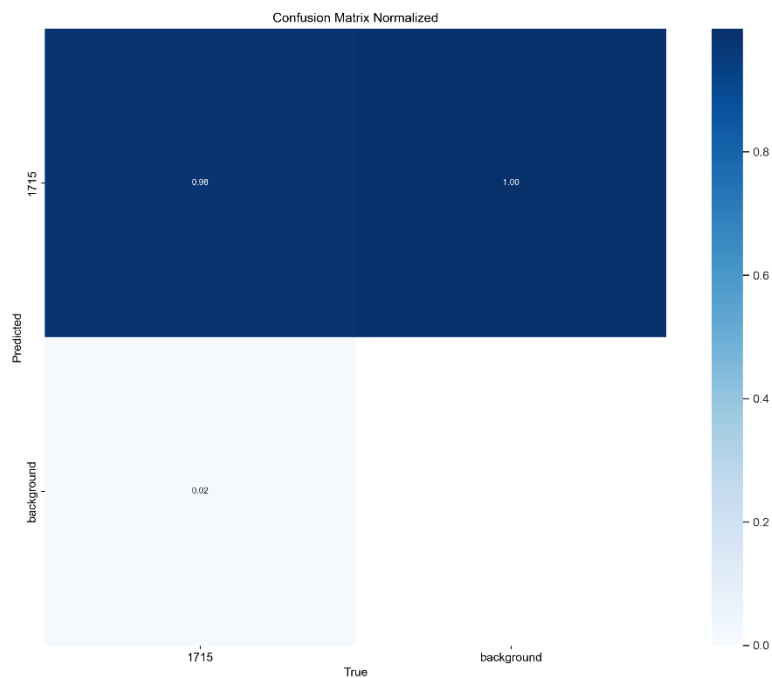


Рис. 3.6. Нормалізована матриця плутанини нейромережі із 13 епохами

У результаті навчання даної нейронної мережі із 13 епохами точність системи склала 0.98 або ж 98%. У поставленій задачі стоїть вимога до системи у 99% точності, тому продовжимо збільшувати кількість епох навчання. Для наочності виведемо графік навчання та тренування на рисунку 3.7.

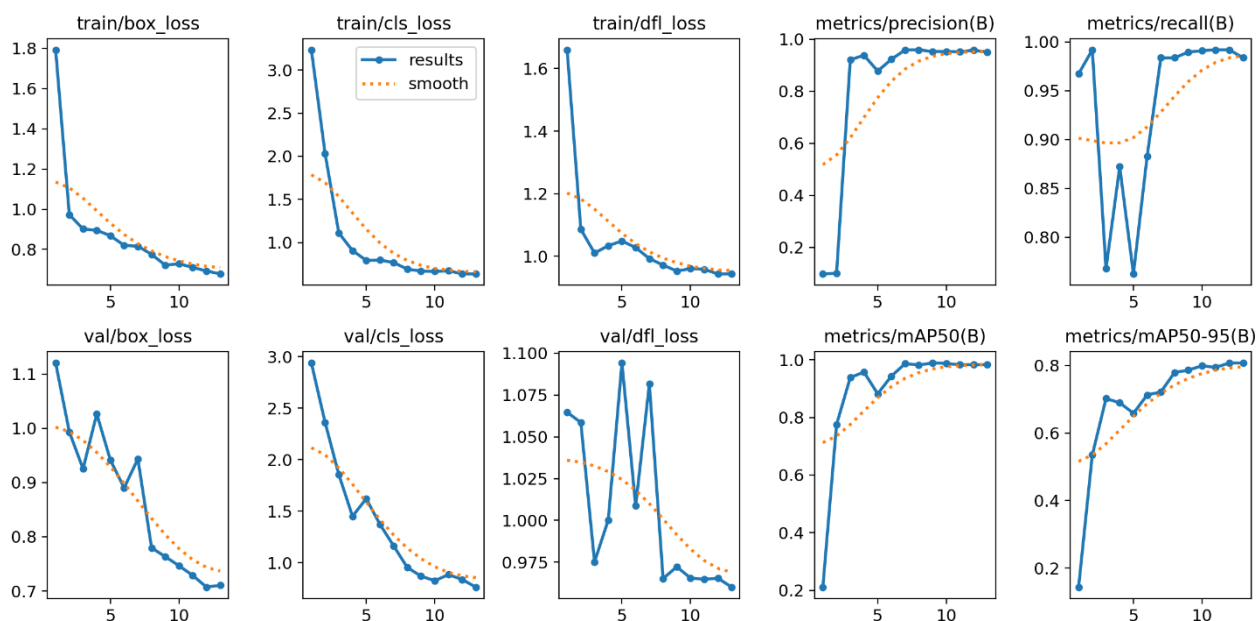


Рис.3.7. Метрики навчання та валідації 13 епох

Наступним етапом спробуємо збільшити крок і почнемо навчання відразу із 20 епохами (рис.3.8).

```

• model = YOLO('yolov8n.pt')
results = model.train(data='dataset.yaml', imgsz=640, epochs=20, batch=16, name='yolov8n_1715')
33.8s Python

Ultralytics YOLOv8.0.227 Python-3.10.4 torch-2.1.2+cpu CPU (Intel Core(TM) i5-9400F 2.90GHz)
engine\trainer: task=detect, mode=train, model=yolov8n.pt, data=dataset.yaml, epochs=20, patience=50, batch=16, imgsz=640, save=Tr
WARNING:tensorflow:From c:\Users\Tapac\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\losses.py:2976: The na

Overriding model.yaml nc=80 with nc=1

      from  n  params module                                arguments
0         -1 1    464 ultralytics.nn.modules.conv.Conv      [3, 16, 3, 2]
1         -1 1   4672 ultralytics.nn.modules.conv.Conv      [16, 32, 3, 2]
2         -1 1   7360 ultralytics.nn.modules.block.C2f      [32, 32, 1, True]
3         -1 1  18560 ultralytics.nn.modules.conv.Conv      [32, 64, 3, 2]
4         -1 2  49664 ultralytics.nn.modules.block.C2f      [64, 64, 2, True]
5         -1 1  73984 ultralytics.nn.modules.conv.Conv      [64, 128, 3, 2]
6         -1 2 197632 ultralytics.nn.modules.block.C2f      [128, 128, 2, True]
7         -1 1 295424 ultralytics.nn.modules.conv.Conv      [128, 256, 3, 2]
8         -1 1 460288 ultralytics.nn.modules.block.C2f      [256, 256, 1, True]
9         -1 1 164608 ultralytics.nn.modules.block.SPPF      [256, 256, 5]
10        -1 1      0 torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
11       [-1, 6] 1      0 ultralytics.nn.modules.conv.Concat      [1]
12        -1 1 148224 ultralytics.nn.modules.block.C2f      [384, 128, 1]
13        -1 1      0 torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
14       [-1, 4] 1      0 ultralytics.nn.modules.conv.Concat      [1]
15        -1 1  37248 ultralytics.nn.modules.block.C2f      [192, 64, 1]
16        -1 1  36992 ultralytics.nn.modules.conv.Conv      [64, 64, 3, 2]
17       [-1, 12] 1      0 ultralytics.nn.modules.conv.Concat      [1]

```

Рис.3.8. Навчання нейромережі із 20 епохами

Аналіз даної версії нейронної мережі розпочнемо за аналогією до попередньої версії з 13-ма епохами, але опустимо детальне пояснення значення кожного графіку. Аналізуватимемо лише основні вихідні дані. Графік F1 confidence наведено на рисунку 3.9.

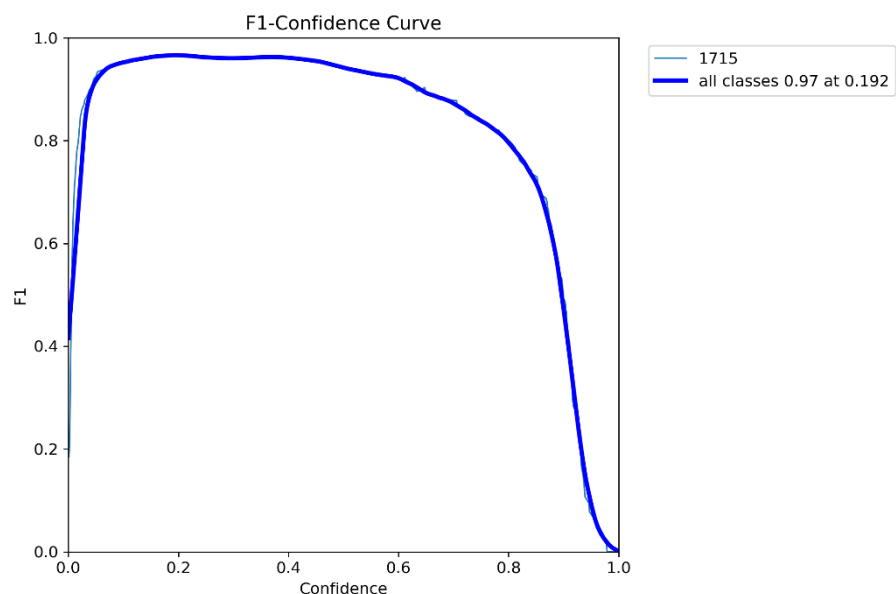


Рис.3.9. F1 confidence з 20-ма епохами навчання

Наступним, за аналогією, на рисунку 3.10. продемонструємо Precision-Confidence Curve .

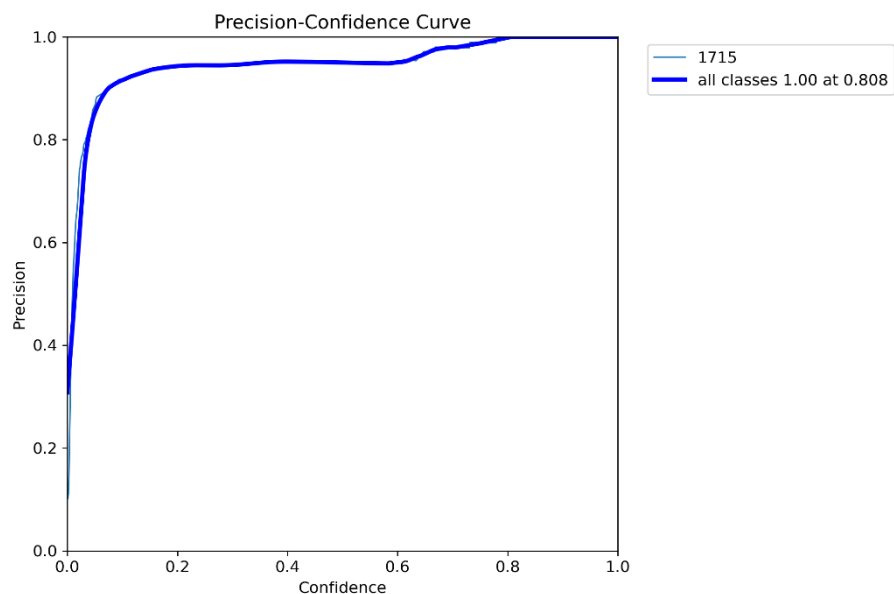


Рис.3.10. Precision-Confidence Curve з 20-ма епохами навчання

Далі, виведемо дві матриці, матрицю плутанини та нормалізовану матрицю плутанини.

Для початку на рисунку 3.11. продемонструємо матрицю плутанини у початковому вигляді.

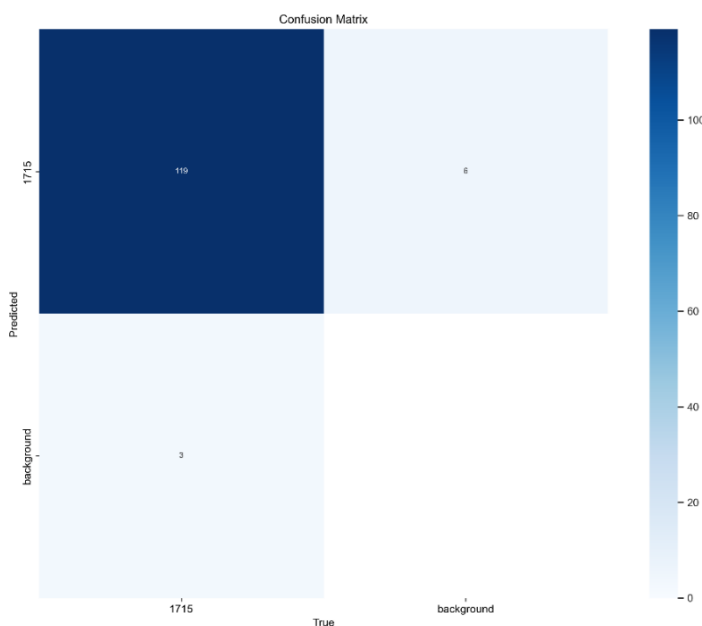


Рис.3.11. Матриця плутанини нейромережі із 20-ма епохами навчання

Демонструємо нормалізовану матрицю плутанини на рисунку 3.12.

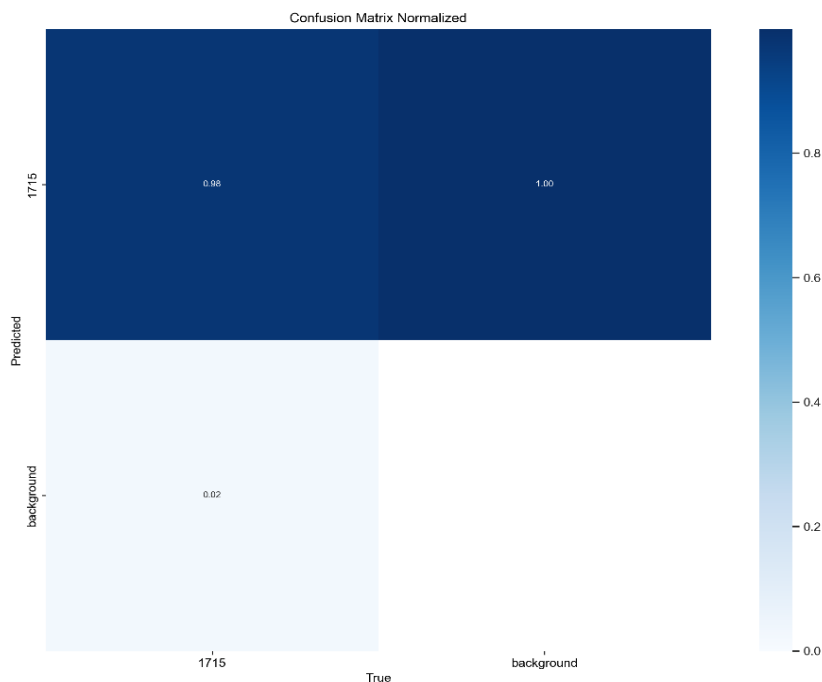


Рис.3.12. Нормалізована матриця плутанини нейромережі із 20-ма епохами навчання

Також, продемонструємо графіки тренування та валідації даних на рисунку 3.13.

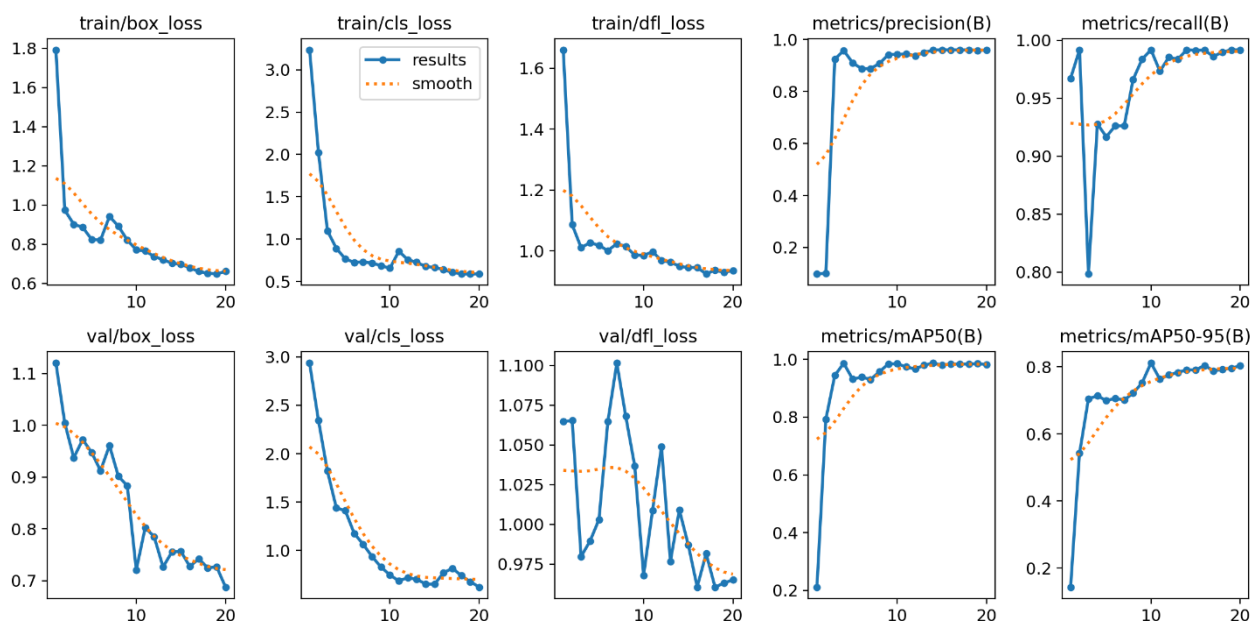


Рис.3.13. Метрики навчання та валідації 20 епох

Виходячи із даних, які вказані на нормалізованій матриці плутанини можна зафіксувати результат у 0.98 або ж 98%. Існує два можливі варіанти подальшого розвитку нейронної мережі, або крок збільшення у 7 епох недостатній для покращення навчання. Або ж навпаки необхідно зменшити кількість епох, тому що при 20 епохах виникло перетренування нейронної мережі. Для подальшого тестування було обрано другий варіант розвитку, а саме зменшення кількості епох до 16.

Повторюємо процес встановлення епох, який показано на рисунку 3.14. У даному випадку встановлюємо кількість епох у розмірі 16 і починаємо тренування.

```

model = YOLO('yolov8n.pt')
results = model.train(data='dataset.yaml', imgsz=640, epochs=16, batch=16, name='yolov8n_1715')
29.8s

Ultralytics YOLOv8.0.227 Python-3.10.4 torch-2.1.2+cpu CPU (Intel Core(TM) i5-9400F 2.90GHz)
engine\trainer: task=detect, mode=train, model=yolov8n.pt, data=dataset.yaml, epochs=16, patience=50, batc
WARNING:tensorflow:From c:\Users\Tapac\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\sr
Overriding model.yaml nc=80 with nc=1

      from  n  params module
0         -1  1      464 ultralytics.nn.modules.conv.Conv [3, 16, 3, 2]
1         -1  1     4672 ultralytics.nn.modules.conv.Conv [16, 32, 3, 2]
2         -1  1     7360 ultralytics.nn.modules.block.C2f [32, 32, 1, True]
3         -1  1    18560 ultralytics.nn.modules.conv.Conv [32, 64, 3, 2]
4         -1  2    49664 ultralytics.nn.modules.block.C2f [64, 64, 2, True]
5         -1  1    73984 ultralytics.nn.modules.conv.Conv [64, 128, 3, 2]
6         -1  2   197632 ultralytics.nn.modules.block.C2f [128, 128, 2, True]
7         -1  1   295424 ultralytics.nn.modules.conv.Conv [128, 256, 3, 2]
8         -1  1   460288 ultralytics.nn.modules.block.C2f [256, 256, 1, True]
9         -1  1   164608 ultralytics.nn.modules.block.SPPF [256, 256, 5]
10        -1  1         0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
11      [-1, 6]  1         0 ultralytics.nn.modules.conv.Concat [1]
12        -1  1   148224 ultralytics.nn.modules.block.C2f [384, 128, 1]
13        -1  1         0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
14      [-1, 4]  1         0 ultralytics.nn.modules.conv.Concat [1]
15        -1  1    37248 ultralytics.nn.modules.block.C2f [192, 64, 1]
16        -1  1    36992 ultralytics.nn.modules.conv.Conv [64, 64, 3, 2]
17      [-1, 12]  1         0 ultralytics.nn.modules.conv.Concat [1]

```

Рис.3.14. Встановлення кількості епох у розмірі 16

Після завершення навчання виводимо відповідні графіки. Перший графік на черзі F1 confidence, який показано на рисунку 3.15.

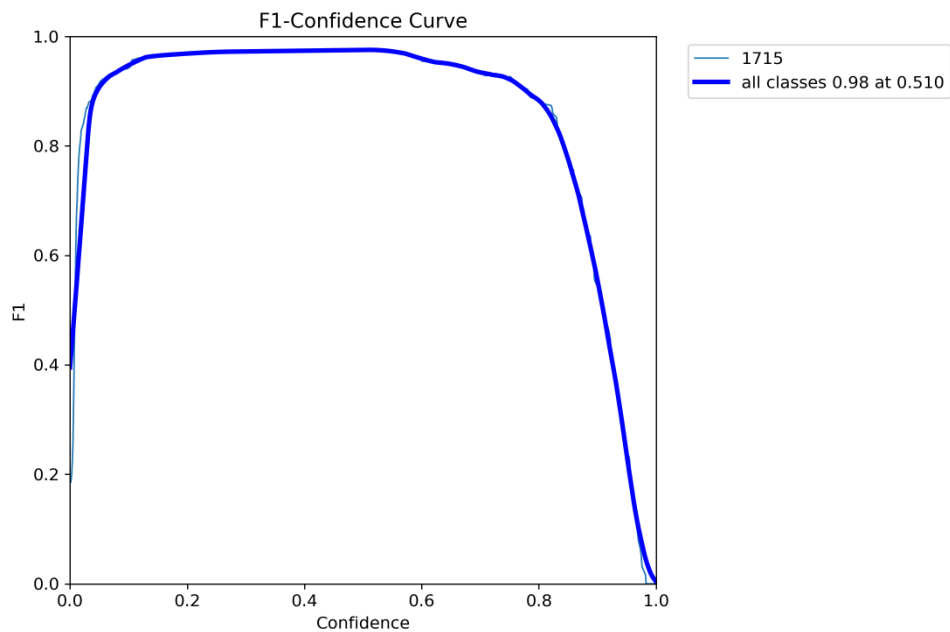


Рис.3.15. F1 confidence із 16 епохами навчання

Далі, на рисунку 3.16 покажемо Precision-Confidence Curve.

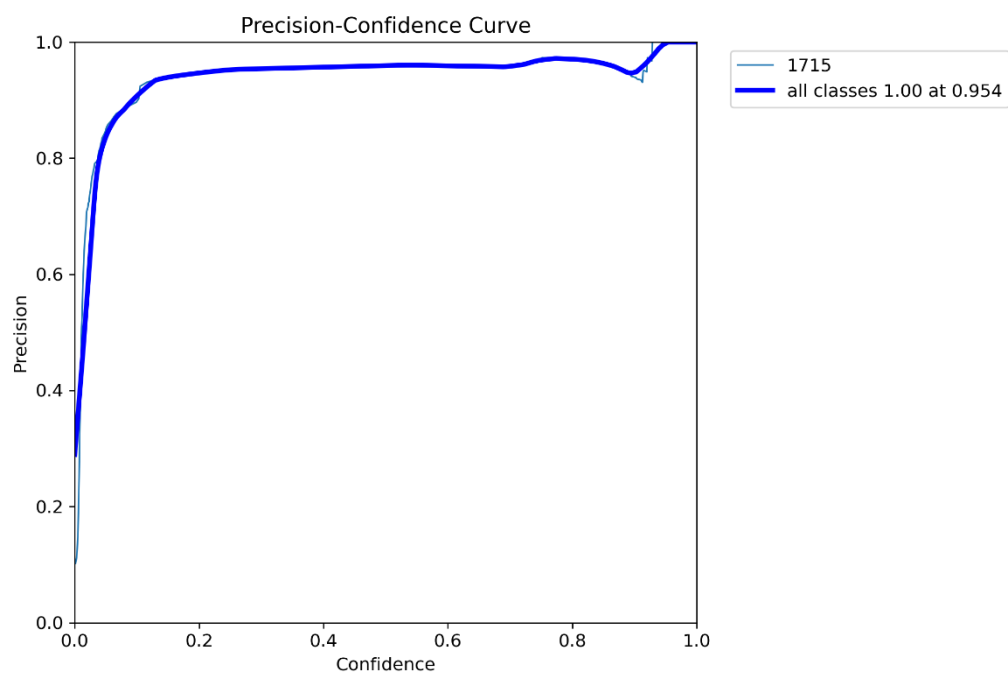


Рис.3.16. Precision-Confidence Curve з навчанням у 16 епох

Наступним кроком продемонструємо нормалізовану матрицю плутанини на рисунку 3.17., оскільки без нормалізації дана матриця не несе для нас цінної інформації.

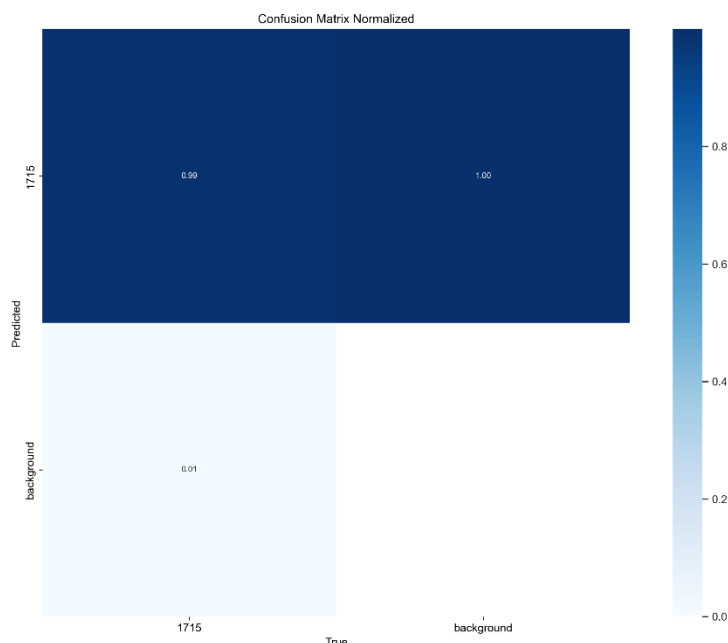


Рис.3.17. Нормалізована матриця плутанини із навчанням у 16 епох

Результат даної версії нейронної моделі із навчанням у 16 епох дав результат 0,99 або ж 99%. Даний результат цілком задовільняє поставлене завдання щодо якості детектування. Порівняння різних версій навчання показано у таблиці 3.2.

Таблица 3.2.

Кількість епох	10	13	16	20
F1 confidence	0,97 at 0,098	0,97 at 0,225	0,98 at 0,510	0,97 at 0,192
Precision- Confidence	1,00 at 0,607	1,00 at 0,819	1,00 at 0,954	1,00 at 0,808
Нормалізована матриця плутанини	0,93	0,98	0,99	0,98

Виходячи із даних, які наведені у таблиці номер 2, можна чітко побачити, як впливає кількість епох на точність системи. Починаючи із найгіршої версії у 10 епох із результатом у 93% до найкращої за точністю версії з 16 епох з результатом у 99%. При збільшенні кількості епох після 16 настає так зване перетренування, коли точність системи падає. Слугувати цьому може

недостатньо якісний датасет, тобто мала кількість даних для тренування, або ж недостатньо якісні дані для валідації чи тесту.

3.2. Розгортання та тестування моделі у вигляді веб-додатку

Для того, щоб розгорнути нейронну модуль у вигляді веб-додатку необхідно перш за все активувати Backend та Frontend частини. Процес створення та відображення даних модулів показаний у другому розділі. Занотуємо основні команди для запуску серверної частини:

1. Створюємо середовище під проект

```
python -m venv venv
```

2. Активувати середовище (після виконання має бути префікс (venv)

біля шляху вводу команди)

```
.\venv\Scripts\activate
```

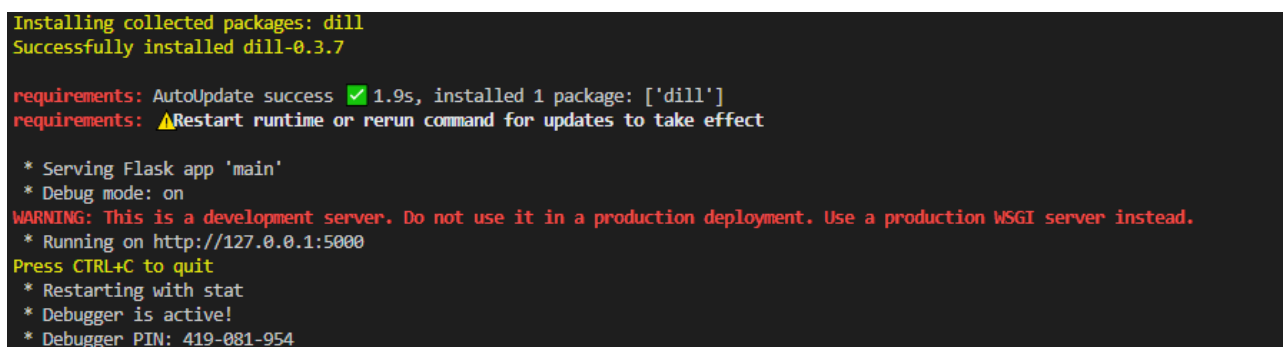
3. Інстал залежностей

```
pip install -r .\requirements.txt
```

4. Старт сервера

```
python main.py
```

На рисунку 3.18. продемонстрований результат успішної активації серверної частини у командному рядку.



```
Installing collected packages: dill
Successfully installed dill-0.3.7

requirements: AutoUpdate success 1.9s, installed 1 package: ['dill']
requirements: ⚠Restart runtime or rerun command for updates to take effect

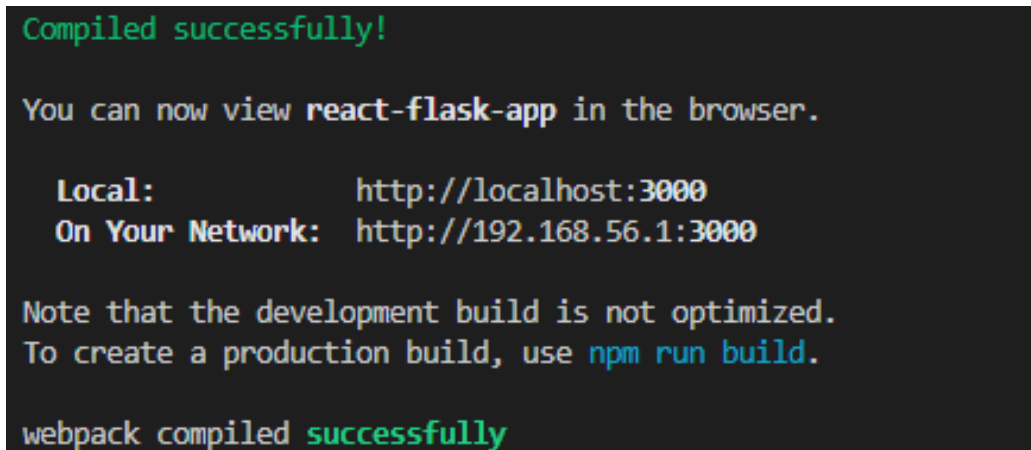
* Serving Flask app 'main'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 419-081-954
```

Рис.3.18. Активація серверної частини

Наступним кроком необхідно активувати клієнтську частину для взаємодії користувача із нейронною мережею. Наведемо приклад команд для запуску клієнтської частини.

1. `cd react-flask-app`
2. `npm i`
3. `npm start`
4. <http://localhost:3000/>

Результат успішного розгортання клієнтської частини показаний на рисунку 3.19.



```
Compiled successfully!  
  
You can now view react-flask-app in the browser.  
  
Local:      http://localhost:3000  
On Your Network:  http://192.168.56.1:3000  
  
Note that the development build is not optimized.  
To create a production build, use npm run build.  
  
webpack compiled successfully
```

Рис.3.19. Успішне розгортання клієнтської частини

Після успішного розгортання нейронної мережі у вигляді веб-додатку перейдемо до її безпосереднього тестування.

Переходимо за адресою <http://localhost:3000/> та натискаємо кнопку «Вибрати файл». Після натискання кнопки у діалоговому вікні обираємо потрібний нам файл із створеного датасету, як показано на рисунку 3.20.

Вибрати файл Файл не вибрано Upload and Detect

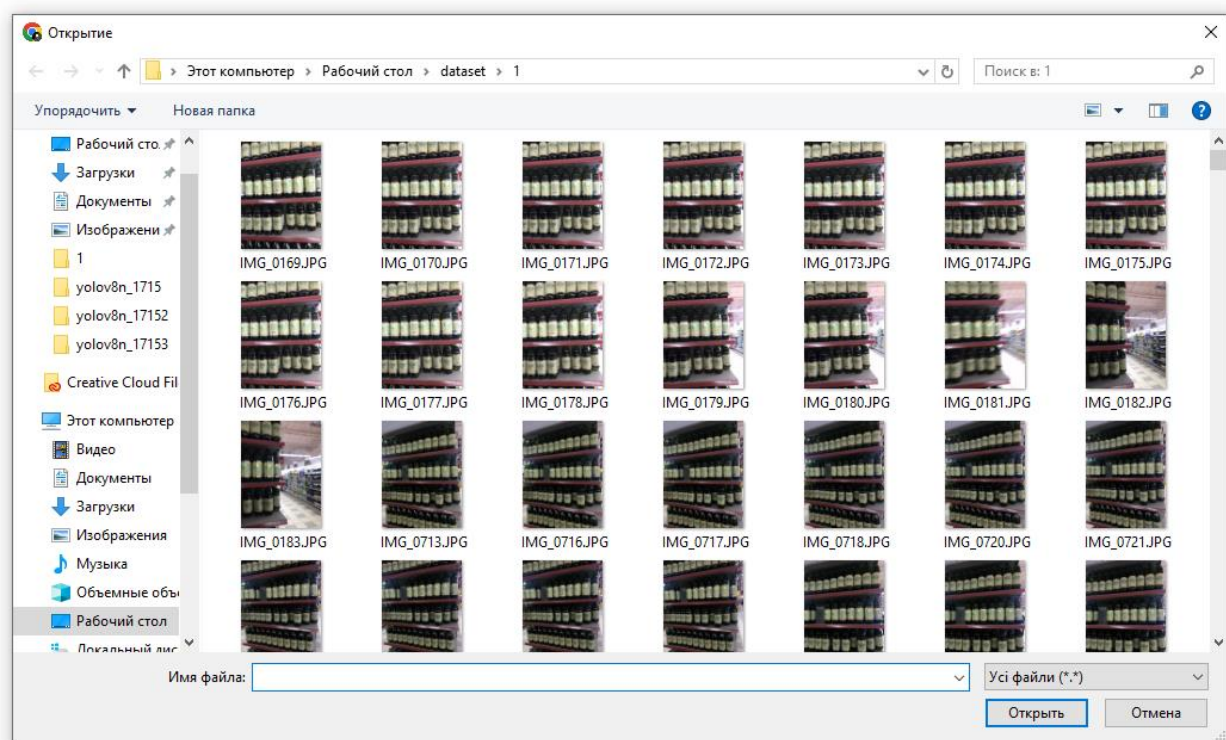


Рис.3.20. Діалогове вікно

Після вибору файлу натискаємо «Відкрити». Наступним кроком натискаємо кнопку «Upload and Detect». Після цього на головному екрані отримуємо результат із задетектованою продукцією, як показано на рисунку 3.21.



Рис.3.22. Тестовий файл №1

На рисунку 3.23. продемонструємо «змазаний» файл і проаналізуємо результат.



Рис.3.23. Тестовий файл №2

Після отримання результатів можна зазначити, що всі одиниці продукції, які були створені у русі, а тому мають розмите зображення також визначені.

Наступним кроком оберемо зображення із затемненням, що буде слугувати додатковою перепорою для визначення об'єктів. Результат детектування буде показано на рисунку 3.24.

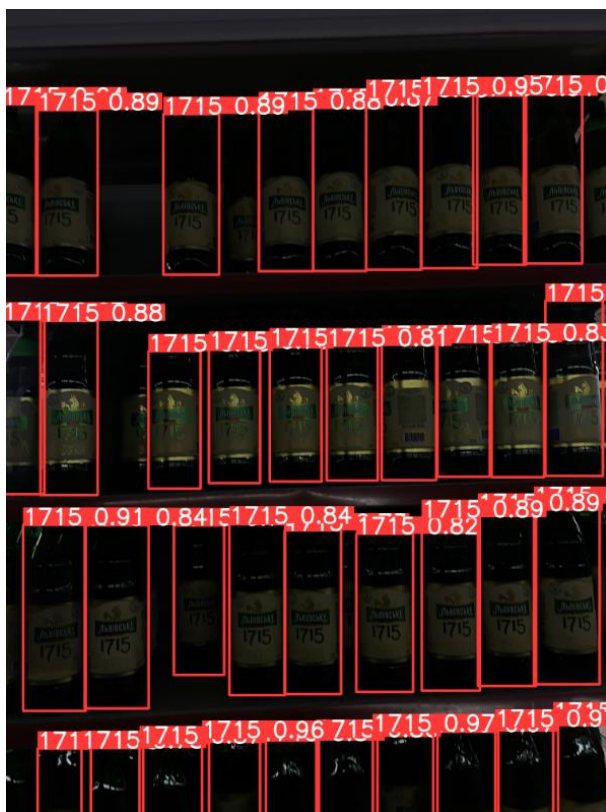


Рис.3.24. Тестовий файл №3

Результати нейронної мережі вражаючі. Людському окові важко ідентифікувати всі одиниці продукції враховуючи затемнення. Без підвищення яскравості це буде важким завданням, однак модель YoloV8n впоралась із поставленим завданням.

Завершальний тестовий зразок буде показано на рисунку 3.25. Результати детектування файлу номер 4 є задовільними. Майже всі об'єкти ідентифіковано, хоча залишається декілька одиниць продукції, які не визначено нейромережею.

Причиною цього може бути декілька причин. Основна – це обмежена кількість навчальних даних. Чим більший датасет для навчання та тренування системи тим відповідно кращий результаті детектування.



Рис.3.25. Тестовий файл №4

Не зважаючи на невелику кількість тренувальних даних (рис.3.26.) розроблена нейронна мережа змогла показати хороший результат у детектуванні продукції та може покращувати результат при збільшенні та покращенні якості датасету для навчання та валідації.

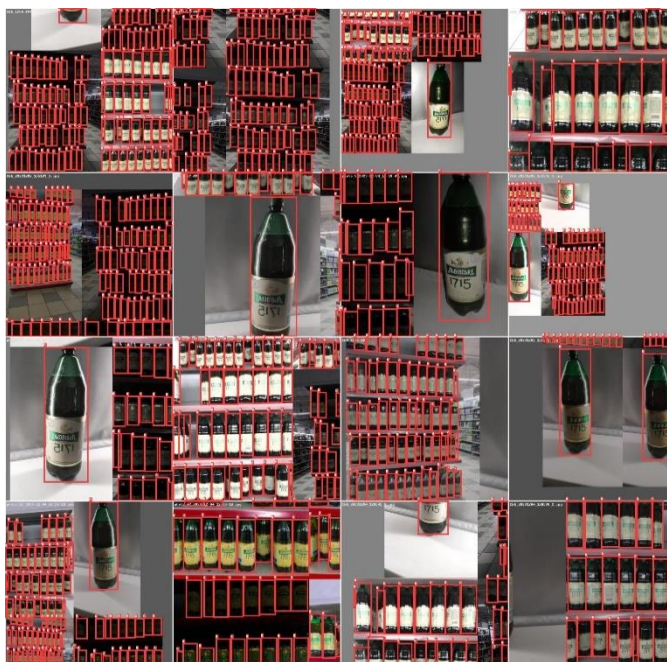


Рис.3.26. Навчальні дані

Розроблена система розпізнавання наявності продукції може стати у нагоду середньому та великому бізнесу у сфері мерчендайзингу. Розроблена неймережа дозволить автоматизувати та пришвидшити роботу із виявлення наявності та кількості продукції на полицях за рахунок роботи з великими обсягами даних та швидкодії детектування.

Висновки до третього розділу

В даному розділі було проведене тестування розробленої нейромережі для детектування об'єктів. Перш за все було навчено системи до максимально можливого позитивного результату, який система показала при 16 епохах навчання. Варто зауважити, що при 20 епохах система погіршила результат у точності виявленні об'єктів. Це може свідчити про перенавчання системи. Можливі причини перенавчання також було наведено у даному розділі.

Показники системи при змінених гіперпараметрах наведено у таблиці 3.2. Також проведено порівняльний аналіз даної таблиці.

На завершальному етапі було розгорнуто створений веб-додаток з інтегрованою системою комп'ютерного зору. Було протестовано дану системи на випадкових тестових даних із різними вхідними параметрами та описано роботу системи по кожному із них. Точність розробленої нейронної мережі детектування об'єктів склала 99%.

ВИСНОВКИ

У кваліфікаційній магістерській роботі було розроблено метод розпізнавання продукції для визначення наявності за допомогою нейронної мережі.

Обґрунтування актуальності теми дало змогу оцінити запровадження систем штучного інтелекту в сфері життєдіяльності людини та доцільність розробки даної системи.

Було проведено основні інструменти та методи розробки даних систем. Після огляд найпопулярніших бібліотек мови програмування Python для машинного навчання та роботи із великими об'ємами даних склалося чітке розуміння напрямку розробки та вибору шляху реалізації.

Наступним, було оглянуто та проаналізовано основні методи навчання штучного інтелекту. Дана інформація дала змогу краще зрозуміти роботи системи із середини. Різні підходи в машинному навчанні дозволяють працювати з різними типами даних та вирішувати різноманітні завдання з прогнозування, класифікації та аналізу. Один із методів краще підходить для вирішення проблем із класифікації, інший для застосування у комп'ютерних іграх. Для поставленого завдання було обрано метод навчання без вчителя.

Оглянувши основні методи навчання, наступним кроком було досліджено найпопулярніші архітектури розпізнавання. Виходячи із інформації поданої у даному підрозділі можна зазначити, що архітектури, як і методи можуть слугувати краще для тих чи інших систем. Деякі краще працюють із текстом, в той час як інші налаштовані для обробки зображень. Для реалізації методу розпізнавання продукції для визначення наявності було обрано архітектуру сімейства YOLO, а саме її останню версію YOLOV8. Будучи передовою, сучасною моделлю, YOLOV8 заснована на успіху попередніх версій, впроваджуючи нові функції та вдосконалення для підвищення продуктивності, гнучкості та ефективності.

Після завершення огляду інформаційних джерел було обрано основні бібліотеки, метод та архітектуру та почато роботу із реалізації даної системи. Перш за все, було створено датасет із даними для навчання та перевірки системи. Наступним кроком було розробка нейронної мережі. Для цього було використано бібліотеку Ultralytics та модель YOLOv8. Після навчання та валідації системи, її було протестовано на незалежних даних та отримано точність системи у 93%.

Після завершення роботи над нейронною мережею для розпізнавання об'єктів було створено архітектуру та розроблено веб-додаток для демонстрації результатів роботи нейронної мережі. Backend частина була реалізована за допомогою фреймворку Flask мови програмування Python. Frontend частина була створена за допомогою бібліотеки React.js.

Завершальним етапом розробки даної системи було тренування нейронної мережі до отримання найкращого можливого результату. Змінюючи гіперпараметри навчання розробленої системи вдалося досягнути результату точності у 99%. Також, порівняно результати попередніх навчань. Розгорнувши дану нейронну мережу у створеному веб-додатку було протестовано працеспроможність та точність системи на незалежних даних. Описано роботу системи при різних вхідних даних.

Дана система розпізнавання продуктів для визначення наявності, що була створена за допомогою нейромережі, може бути корисною для середніх та великих компаній, що займаються мерчендайзингом. Завдяки здатності автоматизувати процес виявлення наявності та кількості продукції на полицях та оперативній обробці великих обсягів даних, ця система може покращити продуктивність та зменшити час на виконання завдань.

Мету кваліфікаційної роботи досягнуто згідно кожного поставленого завдання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Guido S. Introduction to Machine Learning with Python / S. Guido, A. Müller. – Sebastopol, United States: O'Reilly Media, Inc, USA, 2016. – 392 с.
2. Bisong, E., & Bisong, E. (2019). NumPy. Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners, 91-113 с.
3. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, 2825-2830.
4. Molin, Stefanie. Hands-On Data Analysis with Pandas: Efficiently perform data collection, wrangling, analysis, and visualization using Python. Packt Publishing Ltd, 2019.
5. Shukla, Nishant, and Kenneth Fricklas. Machine learning with TensorFlow. Greenwich: Manning, 2018.
6. Gulli, Antonio, and Sujit Pal. Deep learning with Keras. Packt Publishing Ltd, 2017.
7. Jordan, Michael I., and Tom M. Mitchell. "Machine learning: Trends, perspectives, and prospects." Science 349.6245 (2015): 255-260 с.
8. Hastie T. Overview of Supervised Learning / T. Hastie, R. Tibshirani, J. Friedman // The Elements of Statistical Learning / T. Hastie, R. Tibshirani, J. Friedman. – New York: Springer, New York, NY, 2009. – (Springer Нью-Йорк, Нью-Йорк). – (Springer Series in Statistics; 2). – С. 9–41.
9. Мілян, Н. "Аналіз методів машинного навчання з вчителем." Збірник тез Міжнародної студентської науково-технічної конференції „Природничі та гуманітарні науки. Актуальні питання “ 1 (2018): 51-52 с.
10. Капленко, Артем Анатолійович. Дослідження та програмна реалізація методів класифікації зображень на основі навчання з підкріпленням. MS thesis. Національний університет" Запорізька політехніка", 2022.

11. Мельник, Анна, та Даниїл Берестенко. "Дослідження методів машинного навчання." Автоматика, комп'ютерно-інтегровані технології та проблеми енергоефективності в промисловості і сільському господарстві (АКІТ-2022): Матеріали міжнародної науково-технічної конференції.–Кропивницький: ПП «Ексклюзив-Систем», 2022.–243 с.
12. Alzubaidi L. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions / L. Alzubaidi, J. Zhang, A. Humaidi // Journal of Big Data / L. Alzubaidi, J. Zhang, A. Humaidi., 2021. – (8; т. 53).
13. Yuan, Zheng-Wu, and Jun Zhang. "Feature extraction and image retrieval based on AlexNet." Eighth International Conference on Digital Image Processing (ICDIP 2016). Vol. 10033. SPIE, 2016.
14. Anand, R., et al. "Face recognition and classification using GoogleNET architecture." Soft Computing for Problem Solving: SocProS 2018, Volume 1. Springer Singapore, 2020.
15. Gaudenz B. VGG Very Deep Convolutional Networks (VGGNet) – What you need to know [Електронний ресурс] / Boesch Gaudenz // viso.ai – Режим доступу до ресурсу: <https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/>.
16. He, Fengxiang, Tongliang Liu, and Dacheng Tao. "Why resnet works? residuals generalize." IEEE transactions on neural networks and learning systems 31.12 (2020): 5349-5362.
17. Jiang, Peiyuan, et al. "A Review of Yolo algorithm developments." Procedia Computer Science 199 (2022): 1066-1073.
18. Ponnusamy A. YOLO Object Detection with OpenCV and Python [Електронний ресурс] / Arun Ponnusamy // Towards Data Science. – 2018. – Режим доступу до ресурсу: <https://towardsdatascience.com/yolo-object-detection-with-opencv-and-python-21e50ac599e9>.
19. Jocher G. Ultralytics YOLOv8 Docs [Електронний ресурс] / G. Jocher, A. Chaurasia // Ultralytics Inc.. – 2023. – Режим доступу до ресурсу: <https://docs.ultralytics.com/>.

20. Jocher G. Model Training with Ultralytics YOLO [Електронний ресурс] / Glenn Jocher // Ultralytics Inc.. – 2023. – Режим доступу до ресурсу: <https://docs.ultralytics.com/modes/train/>.
21. Sören E. Frontend vs. Backend vs. Full-Stack – The Difference Explained [Електронний ресурс] / Elser Sören // ElevateX GmbH.. – 2022. – Режим доступу до ресурсу: <https://elevatex.de/blog/it-insights/frontend-vs-backend-vs-fullstack-differences/>.
22. Getting started with React [Електронний ресурс] // MDN contributors. – 2023. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started.
23. Flask's documentation [Електронний ресурс] // Pallets. – 2010. – Режим доступу до ресурсу: <https://flask.palletsprojects.com/en/3.0.x/>.
24. Ronacher A. Flask-CORS [Електронний ресурс] / Armin Ronacher // Cory Dolphin Revision. – 2013. – Режим доступу до ресурсу: <https://flask-cors.readthedocs.io/en/latest/>.
25. Lebedzinski P. A Single Number Metric for Evaluating Object Detection Models [Електронний ресурс] / Peter Lebedzinski // Towards Data Science. – 2021. – Режим доступу до ресурсу: <https://towardsdatascience.com/a-single-number-metric-for-evaluating-object-detection-models-c97f4a98616d>.
26. Кравченко С.М., Предчук Т.В. Пошук зображень за допомогою комп'ютерного зору. Тези VI Всеукраїнської науково-технічної конференції здобувачів вищої освіти і молодих учених «Комп'ютерні технології: інновації, проблеми, рішення», 29-30 листопада 2023 року. Житомир: "Житомирська політехніка", 2023 (подано до друку).
27. Кравченко С.М., Предчук Т.В. Згорткова нейронна мережа для розпізнавання об'єктів. Тези VI Всеукраїнської науково-технічної конференції здобувачів вищої освіти і молодих учених «Комп'ютерні технології: інновації, проблеми, рішення», 29-30 листопада 2023 року. Житомир: "Житомирська політехніка", 2023 (подано до друку).

ДОДАТКИ

Лістинг файлу Train.ipynb

```
!pip install ultralytics
from ultralytics import YOLO
from IPython.display import display, Image
import glob
model = YOLO('yolov8n.pt')
results = model.train(data='dataset.yaml', imgsz=640, epochs=16, batch=16,
name='yolov8n_1715')
model = YOLO('runs/detect/yolov8n_1715/weights/best.pt')
result = model.predict('./datasets/test', save=True, conf=0.5)
for image_path in glob.glob(f'{result[0].save_dir}/*.jpg'):
    display(Image(filename=image_path, width=600))
```

Лістинг файлу App.js

```

import React, { useState } from 'react';
import axios from 'axios';

function App() {
  const [selectedFile, setSelectedFile] = useState(null);
  const [imageSrc, setImageSrc] = useState('');

  const handleFileChange = (event) => {
    setSelectedFile(event.target.files[0]);
  };

  const handleUpload = async () => {
    const formData = new FormData();
    formData.append('file', selectedFile);

    try {
      const response = await axios.post('http://localhost:5000/detect', formData, {
        responseType: 'arraybuffer' // change to arraybuffer to handle binary data
      })
      const base64 = btoa(
        new Uint8Array(response.data).reduce(
          (data, byte) => data + String.fromCharCode(byte),
          ''
        ),
      );
      setImageSrc(`data:image/jpeg;base64,${base64}`);
    } catch(error) {
      console.error('Error uploading the file', error);
    }
  };

  return (
    <div className="App">
      <input type="file" onChange={handleFileChange} />
      <button onClick={handleUpload}>Upload and Detect</button>
      {imageSrc && <img src={imageSrc} alt="Detected" style={{ maxWidth: '1200px',
maxHeight: '800px' }} />}
    </div>
  );
}

export default App;

```

Лістинг файлу main.py

```
import json

from flask import Flask, request, send_file
from flask_cors import CORS
from ultralytics import YOLO
import os
import tempfile
import glob

app = Flask(__name__)

CORS(app)

model_path = 'runs/detect/yolov8n_1715/weights/best.pt'
model = YOLO(model_path)

@app.route('/detect', methods=['POST'])
def detect():
    if 'file' not in request.files:
        return "No file part", 400

    file = request.files['file']
    if file.filename == '':
        return "No selected file", 400

    if file:
        # Використовуємо tempfile для створення тимчасового файла
        with tempfile.NamedTemporaryFile(delete=False, suffix='.jpg') as tmp:
            file.save(tmp.name)
            # Виконуємо виявлення об'єктів
            result = model.predict(tmp.name, save=True, conf=0.5)

        # Видаляємо тимчасовий файл
        os.remove(tmp.name)

        image_path = os.path.join(os.path.curdir, result[0].save_dir,
os.path.basename(result[0].path))

        image = glob.glob(image_path)[0]

        return send_file(image, mimetype='image/jpeg', as_attachment=True)

if __name__ == '__main__':
    app.run(debug=True)
```