МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувачка кафедри,
д-р техн. наук, проф.

_____Ірина ЖУРАВСЬКА

« __ » _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

РОЗПОДІЛЕНА СИСТЕМА HEALTH-МОНІТОРИНГУ ТЕПЛИЦЬ

Спеціальність 123 Комп'ютерна інженерія

Освітня програма «Комп'ютерна інженерія»

Здобувач _____Дмитро КИСЕЛЬОВ

«___»_____ 2024 р.

Керівник канд. фіз.-мат. наук, доцент _____ Сергій ПУЗИРЬОВ

«___»_____ 2024 р.

Миколаїв – 2024

Чорноморський національний університет імені Петра Могили

(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Комп’ютерної інженерії
Рівень вищої освіти	Другий (магістерський)
Освітній ступень	Магістр
Спеціальність	123 Комп’ютерна інженерія
Освітня програма	Комп’ютерна інженерія

ЗАТВЕРДЖУЮ

Завідувачка кафедри,
д-р техн. наук, проф.

_____ Ірина ЖУРАВСЬКА
«___»_____2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

_____Кисельова Дмитра Максимовича_____
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи
Розподілена система health-моніторингу теплиць_____

Затверджена наказом ректора ЧНУ ім. Петра Могили від___№___.

2. Строк представлення кваліфікаційної роботи «____» _____ 20__ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні
___Очікуваним результатом роботи є апаратно-програмний комплекс для розпізнавання хвороб рослин у теплицях, що дозволило б покращити якість покращити якість догляду за врожаєм, збільшити ефективність вирощування. _____

4. Перелік питань, що підлягають розробці:

1) огляд сучасних методів для розпізнавання образів та засобів для виявлення хвороб у рослинан ;

2) аналіз переваг та недоліків існуючих систем;

3) розробити апаратну частину розподіленої системи, яка складається з одноплатного комп'ютера та системи відеоспостереження для передачі даних до нейромережі ;

4) розробити програмну частину комплексу, яка складається з тренування нейромережі для розпізнавання рослин та системи оповіщення.

5. Перелік графічних матеріалів

_слайди презентації

6. Завдання до спеціальної частини _

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи

Керівник роботи _____ _ Сергій ПУЗИРЬОВ ___
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Здобувач _____ Дмитро КИСЕЛЬОВ
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Дата видачі завдання « ____»_____20_____р

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної магістерської роботи

Тема:_ Розподілена система health-моніторингу теплиць

№ з/п	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КМР	01.09.2024	16.09.2024	*Виконано*
2.	Огляд літератури за темою роботи	17.09.2024	22.09.2024	*Виконано*
3.	Складання календарного плану КМР	23.09.2024	25.09.2024	*Виконано*
4.	Аналіз предметної області	26.09.2024	29.09.2024	*Виконано*
5.	Розробка проєктних рішень	30.09.2024	07.10.2024	*Виконано*
6.	Моделювання	08.10.2024	20.10.2024	*Виконано*
7.	Оформлення КМР та презентації	21.10.2024	29.10.2024	*Виконано*
8.	Перший передзахист КМР	30.10.2024	31.10.2024	*Виконано*
9.	Конструювання АПК	01.11.2024	15.11.2024	*Виконано*
10.	Розробка програмного забезпечення	16.11.2024	23.11.2024	*Виконано*
11.	Аналіз результатів тестування	25.11.2024	27.11.2024	*Виконано*
12.	Другий передзахист КМР	28.11.2024	29.11.2024	*Виконано*
13.	Відгук керівника КМР	30.11.2024	03.12.2024	*Виконано*
14.	Завершення оформлення КМР та презентації	04.12.2024	09.12.2024	*Виконано*
15.	Рецензування	10.12.2024	13.12.2024	*Виконано*
16.	Захист кваліфікаційної роботи	19.12.2024	20.12.2024	*Виконано*

Керівник роботи _____ __Сергій ПУЗИРЬОВ _____
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Здобувач _____ Дмитро КИСЕЛЬОВ
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

АНОТАЦІЯ

до кваліфікаційної магістерської роботи

«Розподілена система health-моніторингу теплиць»

Здобувач гр. 605м: Кисельов Дмитро Максимович

Керівник: канд. фіз.-мат. наук. доц. Пузирьов С. В.

У даній роботі проведено дослідження і розробку розподіленої системи health-моніторингу для теплиць, яка використовує нейронні мережі для автоматичного розпізнавання хвороб рослин. Зростаючий попит на автоматизовані рішення для моніторингу стану рослин в умовах змін клімату та розвитку сільського господарства робить цю тему надзвичайно актуальною. Об'єктом дослідження є процес моніторингу стану рослин в теплицях за допомогою нейронних мереж, а предметом – система розпізнавання хвороб рослин на основі зображень, отриманих за допомогою відеоспостереження.

Метою роботи є підвищення ефективності моніторингу рослин за допомогою розробки і впровадження автоматизованої системи health-моніторингу на основі нейронних мереж. Практична значимість дослідження полягає в оптимізації процесів виявлення захворювань на ранніх етапах, що дозволяє знизити втрати врожаю та покращити якість догляду за рослинами.

Результати роботи можуть бути використані для автоматизації моніторингу стану рослин у тепличних господарствах та агротехнологічних проектах, пов'язаних із моніторингом сільськогосподарських культур. Робота включає розробку архітектури розподіленої системи, реалізацію нейронної мережі для розпізнавання хвороб рослин, а також впровадження алгоритмів обробки та передачі даних у розподіленій системі.

Вступ містить основні обґрунтування актуальності розробки обраної теми, об’єкт, предмет дослідження, мету та завдання які необхідно виконати для досягнення поставленої мети.

В першому розділі проведено аналіз різноманітних методів, засобів та аналогів систем моніторингу та розпізнавання об’єктів. Розроблено специфікацію вимог до апаратно-програмного комплексу.

В другому розділі міститься опис процесу проєктування апаратно-програмного комплексу.

Третій розділ містить результати апаратно-програмної частини АПК.

У четвертому розділі проведено експериментальні дослідження (тестування, перевірка працездатності АПЗ), виконано аналіз результатів розробки.

Робота складається з [69] сторінок (без додатків), [0] таблиць, [34] рисунків та [2] додатки. У процесі дослідження було використано [25] джерел.

***Ключові слова**: моніторинг, розподілена система, розпізнавання хвороб, спостереження, Raspberry Pi, TensorFlow, навчання моделей.*

ABSTRACT

of the Master's Thesis

«Distributed health monitoring system for greenhouses»

Applicant.: Dmytro Maksymovych Kyselov

Supervisor: Cand. Sci. (Phys.-Math.), Docent Puzyrov S. V.

In this paper, we study and develop a distributed health monitoring system for greenhouses that uses neural networks to automatically recognize plant diseases. The growing demand for automated solutions for plant health monitoring in the context of climate change and agricultural development makes this topic extremely relevant. The object of research is the process of monitoring the condition of plants in greenhouses using neural networks, and the subject is a plant disease recognition system based on images obtained through video surveillance.

The aim of the study is to improve the efficiency of plant monitoring by developing and implementing an automated health monitoring system based on neural networks. The practical significance of the study is to optimize the processes of detecting diseases at early stages, which reduces crop losses and improves the quality of plant care.

The results of the work can be used to automate plant health monitoring in greenhouses and agrotechnological projects related to crop monitoring. The work includes the development of a distributed system architecture, the implementation of a neural network for plant disease recognition, and the implementation of algorithms for data processing and transmission in a distributed system.

The introduction contains the main justifications for the relevance of the chosen topic, the object, subject of research, the goal and tasks to be performed to achieve the goal.

The first chapter analyses various methods, tools and analogues of object monitoring and recognition systems. The specification of requirements for the hardware and software system is developed.

The second section describes the design process of the hardware and software system.

The third section contains the results of the hardware and software part of the AIC.

The fourth section conducts experimental studies (testing, verification of the APS performance) and analyses the development results.

The work consists of [69] pages (without appendices), [0] tables, [34] figures, [N] references and [25] appendices.

***Keywords**: monitor, distributed system, disease recognition, observation, Raspberry Pi, TensorFlow, model training.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

АПК – апаратно-програмний комплекс

ОС – операційна система

ПЗ – програмне забезпечення

ШНМ – штучні нейронні мережі

ANN – Artificial Neural Networks

AR – Augmented reality

ARM – Advanced RISC Machine

GPIO – General-Purpose Input/Output

NMS – non-Maximum Suppression

YOLO – You Only Look Once

ВСТУП

В умовах стрімкого розвитку сільського господарства та глобальних змін клімату, забезпечення ефективного моніторингу стану рослин стає критично важливим завданням для підвищення врожайності та запобігання втратам, спричиненим захворюваннями. Традиційні методи діагностики хвороб рослин зазвичай вимагають участі фахівців та регулярного огляду, що є не лише трудомістким, але й недостатньо ефективним для великих тепличних комплексів. З розвитком технологій штучного інтелекту, зокрема нейронних мереж, відкриваються нові можливості для автоматизації цих процесів.

Одним із перспективних напрямів вирішення проблеми є розробка розподілених систем health-моніторингу теплиць, які поєднують в собі здатність аналізувати зображення рослин в реальному часі з використанням комп'ютерного зору та нейронних мереж для виявлення можливих ознак захворювань. Такий підхід дозволяє не тільки своєчасно виявляти хвороби, але й автоматично відправляти сигнали для коригуючих дій, що знижує залежність від людського фактору.

Об’єктом дослідження є процес моніторингу за допомогою нейронних мереж

Предметом дослідження є система розпізнавання хвороб рослин на основі нейронних мереж.

Метою даної кваліфікаційної магістерської роботи є підвищення ефективності моніторингу стану рослин у теплицях шляхом розробки та впровадження розподіленої системи health-моніторингу на основі нейронних мереж. Це дозволить автоматизувати процес виявлення рослинних захворювань та своєчасно реагувати на зміни у їх стані, що в свою чергу сприятиме зменшенню втрат врожаю та оптимізації догляду за рослинами.

Для досягнення визначеної мети необхідно вирішити такі **завдання**:

– провести аналіз існуючих методів автоматизованого моніторингу розпізнавання образів;

– розробити архітектуру розподіленої системи, що забезпечує безперервне спостереження за станом рослин з використанням відеокамер та нейронних мереж;

– реалізувати нейронну мережу для автоматичного розпізнавання хвороб рослин на основі зображень, отриманих із відеоспостереження;

– впровадити алгоритми обробки та передачі даних у розподіленій системі для своєчасного виявлення та оповіщення про можливі загрози для рослин;

– тестування ефективність розробленої системи.

Обґрунтування основних проєктних рішень. Основні проєктні рішення базуються на використанні розподіленої системи, яка включає відеоспостереження з високою роздільною здатністю, нейронну мережу для розпізнавання хвороб та централізовану обробку даних для швидкого реагування. Такий підхід дозволяє інтегрувати технології комп'ютерного зору та штучного інтелекту в практичні агротехнологічні системи.

Сфера застосування результатів. Результати роботи можуть бути застосовані у тепличних господарствах для автоматизації процесу моніторингу рослин, зокрема для виявлення хвороб на ранніх стадіях розвитку. Система може бути також адаптована для інших агротехнологічних проєктів, що пов’язані з моніторингом стану сільськогосподарських культур.

Дана кваліфікаційна робота була апробована на XXVII Всеукраїнській науково-практичній конференції «МОГИЛЯНСЬКІ ЧИТАННЯ – 2024: досвід та тенденції розвитку суспільства в Україні: глобальний, національний та регіональний аспекти».

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ СИСТЕМИ МОНІТОРИНГУ. ФОРМУВАННЯ ВИМОГ ДО АПАРАТНО-ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Сучасні програмні рішення для розпізнавання образів

Тема комп'ютерного зору є відносно новою і надзвичайно різноманітною. Хоча перші дослідження в цій сфері почали з'являтися значно раніше, інтенсивне вивчення цієї проблеми почалося лише з кінця 1970-х років, коли комп'ютери стали здатні ефективно обробляти великі обсяги даних, зокрема зображення. Однак слід зазначити, що ці дослідження часто виникали у межах інших наукових дисциплін, що ускладнило формування єдиного стандартного визначення проблеми комп'ютерного зору. Крім того, навіть важливішим є те, що не існує універсального підходу до розв'язання цієї проблеми. Замість єдиного рішення існує велика кількість методів, що застосовуються для вирішення конкретних, чітко визначених задач комп'ютерного зору. Ці методи, як правило, спеціалізовані та залежні від конкретного типу задачі, і рідко коли їх можна застосувати для більш широкого спектра завдань [1].

Багато методів і застосувань комп'ютерного зору все ще знаходяться на етапі фундаментальних досліджень, проте дедалі більше з них починають інтегрувати у комерційні продукти. У таких випадках вони часто входять до складу складних систем, здатних вирішувати різноманітні й складні завдання, зокрема у сферах медичних зображень, де важлива точність діагностики, а також у процесах вимірювання і контролю якості на виробництві, де потрібні високі стандарти ефективності та надійності. У більшості практичних застосувань комп'ютерного зору комп'ютери заздалегідь програмуються для виконання конкретних завдань, використовуючи спеціалізовані алгоритми. Однак методи, засновані на знаннях і штучному інтелекті, все більше стають універсальними та здатні адаптуватися до нових ситуацій і задач, що розширює їх застосування у різних сферах і робить системи більш гнучкими та ефективними.

Комп'ютерний зір – це галузь досліджень, яка зосереджена на розробці технологій, що допомагають комп'ютерам «бачити» та інтерпретувати навколишній світ (рис. 1.1). Це сфера, яка в широкому сенсі є підгалуззю штучного інтелекту та машинного навчання і передбачає використання як спеціалізованих методів, так і загальних алгоритмів навчання [2].

У більшості випадків, коли людина сприймає явища навколишнього середовища, вона здійснює їх класифікацію – розподіл на групи за ознаками схожості. Це означає, що об'єкти чи ситуації, хоча і можуть мати істотні відмінності, об'єднуються в одну категорію на основі певних спільних характеристик. Це важлива властивість, яка дозволяє людині адаптивно реагувати на різноманітні ситуації, використовуючи загальні концепції для спрощення розуміння складної інформації.

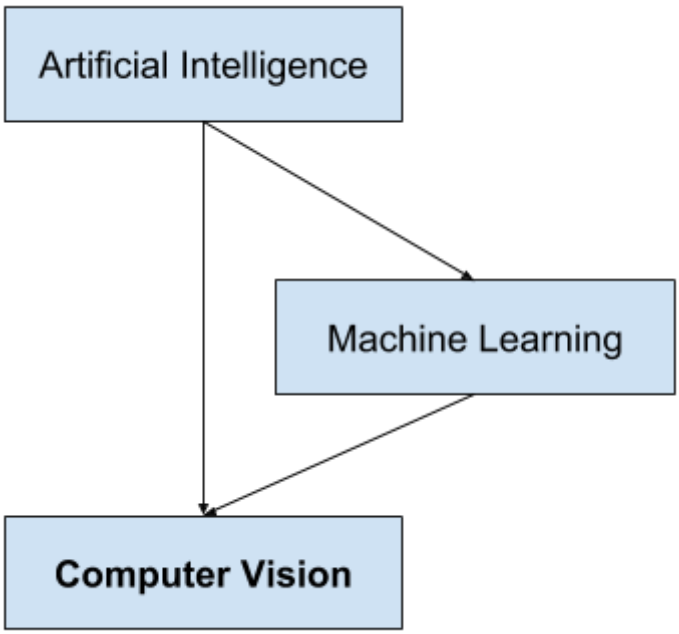


Рисунок. 1.1 – Схема взаємозв'язку між штучним інтелектом та комп'ютерним зором

Для оптичного розпізнавання образів використовують методи, що передбачають обробку зображень об'єкта під різними кутами, масштабами та зсувами, що дозволяє забезпечити його правильне виявлення в різних умовах..

Другий підхід передбачає виявлення контуру об'єкта та аналіз його властивостей, таких як зв’язність і наявність кутів. Цей метод дозволяє детально досліджувати форму і структуру об'єкта для подальшої його класифікації та розпізнавання.

Ще одним підходом є використання штучних нейронних мереж, таких як багатошарові персептрони, мережі квантування, карти Кохонена та рекурентні мережі. Цей метод потребує великої кількості прикладів для навчання (з правильними мітками) або спеціальної архітектури нейронної мережі, яка враховує специфічні особливості задачі, що розв'язується.

Реалізація систем комп'ютерного зору значною мірою залежить від області їхнього застосування. Деякі системи є автономними та спеціалізуються на вирішенні конкретних завдань, таких як виявлення об'єктів чи проведення вимірювань. Інші ж системи входять до складу більших комплексів, що включають підсистеми контролю механічних маніпуляторів, планування, інформаційні бази даних, інтерфейси людина-машина та інші елементи, забезпечуючи комплексне виконання складних завдань.

Реалізація систем комп'ютерного зору також залежить від того, чи має вона заздалегідь визначену функціональність, чи здатна вивчати та змінювати деякі свої частини під час експлуатації. Проте існують функції, що є типовими для більшості систем комп'ютерного зору.

Один з основних етапів – отримання зображень. Цифрові зображення можуть бути отримані за допомогою одного або кількох датчиків зображень. До таких датчиків відносяться різноманітні світлочутливі камери, датчики відстані, радари, ультразвукові сенсори тощо. Залежно від типу датчика, дані, що отримуються, можуть бути у вигляді звичайного 2D-зображення, 3D-зображення чи послідовності зображень. Значення пікселів, як правило, відповідають інтенсивності світла в одній або кількох спектральних смугах (кольорові зображення чи зображення у відтінках сірого). Однак, у деяких випадках, значення пікселів можуть відображати інші фізичні властивості, такі як глибина, поглинання

чи відбиття звукових або електромагнітних хвиль, а також дані, отримані за допомогою ядерного магнітного резонансу [3].

Попередня обробка: перед застосуванням методів комп'ютерного зору до відеоданих з метою вилучення корисної інформації необхідно провести попередню обробку, щоб дані відповідали вимогам, визначеним конкретним методом. Приклади таких обробок включають:

– повторну вибірку для перевірки коректності координатної системи зображення;

– видалення шумів, щоб усунути спотворення, спричинені датчиком;

– покращення контрастності, що дозволяє виявити потрібну інформацію;

– масштабування для покращення розрізнення структур на зображенні.

Відокремлення деталей: на цьому етапі з відеоданих виділяються деталі різного рівня складності. Типові приклади таких деталей:

– лінії та межі;

– локалізовані точки інтересу, такі як кути, краплі або ключові точки. Складніші деталі можуть включати структуру, форму чи рух.

Детектування або сегментація: на цьому етапі приймається рішення, які точки або ділянки зображення є суттєвими для подальшої обробки. Прикладами таких деталей є:

– виділення конкретного набору точок, що є цікавими для аналізу;

– сегментація однієї чи кількох ділянок зображення, що містять характерні об'єкти.

Високорівнева обробка: на цьому етапі вхідні дані зазвичай представляють собою невеликий набір інформації, наприклад, набір точок або ділянка зображення, де, ймовірно, розташований певний об'єкт . Прикладами високорівневої обробки є:

– перевірка того, що дані відповідають умовам, специфічним для конкретного методу чи застосування;

– оцінка ключових параметрів, таких як розташування, розмір чи орієнтація об'єкта;

– класифікація виявленого об'єкта за різними категоріями чи класами для подальшого аналізу або використання.

Відомі наступні програмні продукти, які використовуються для розпізнавання зображень:

– FineReader: програма для оптичного розпізнавання символів, розроблена компанією ABBYY (рис. 1.2);

– YOLO (You Only Look Once): це алгоритм для розпізнавання об'єктів на зображеннях та відео.

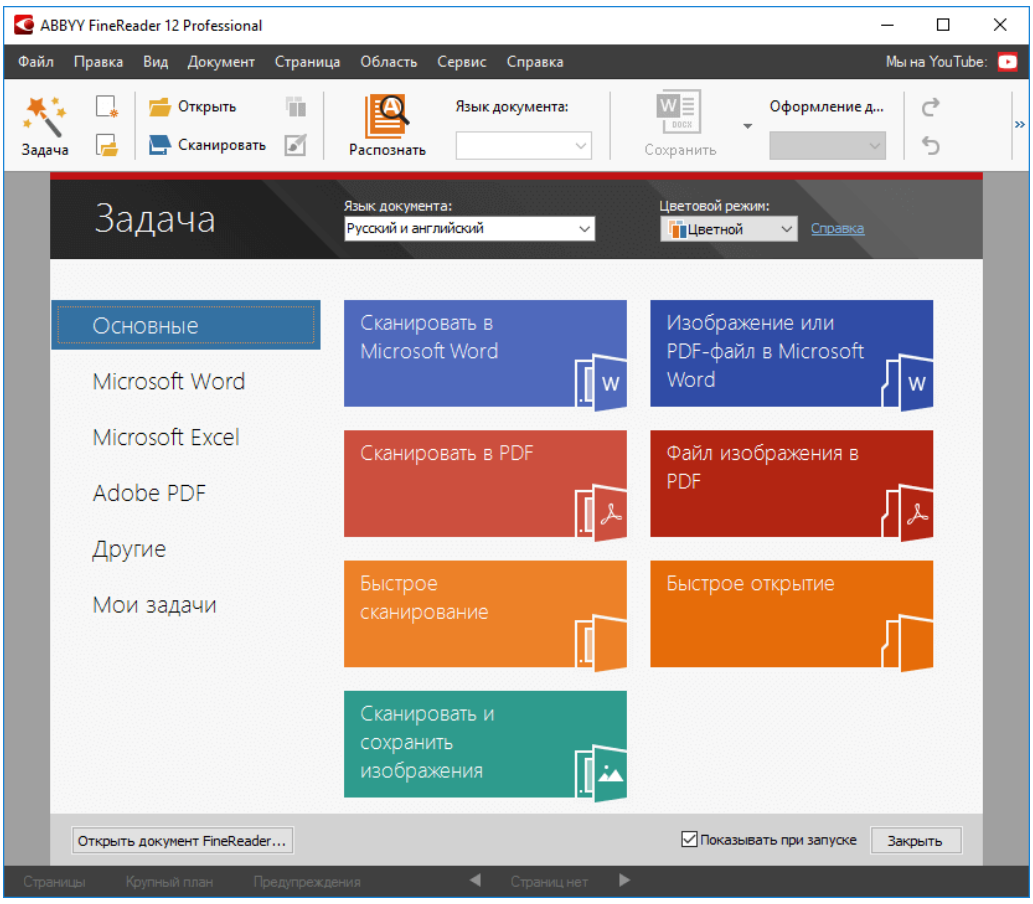


Рисунок 1.2 – Головне вікно програми FineReader

FineReader швидко та з високоточно розпізнає сфотографовані або скановані документи, перетворюючи їх на редагуючі електронні формати або PDF-файли з можливістю пошуку. У разі роботи з якісними документами режим «швидке розпізнавання» дозволяє збільшити швидкість процесу на 40 % без втрати точності. Для чорно-білих документів доступний спеціальний чорно-білий режим, який додатково прискорює обробку на 30 %. FineReader зберігає вихідну структуру

документів які містять багато сторінок, включаючи розташування тексту, таблиць, колонтитулів, приміток, нумерацію сторінок, зміст і багато іншого. Програма підтримує розпізнавання документів на 190 мовах, у будь-яких комбінаціях, а також можливість збереження у найпопулярніших форматах електронних книг, що забезпечує легке створення електронної копії для портативних пристроїв, таких як електронні книги, планшети, смартфони тощо [4].

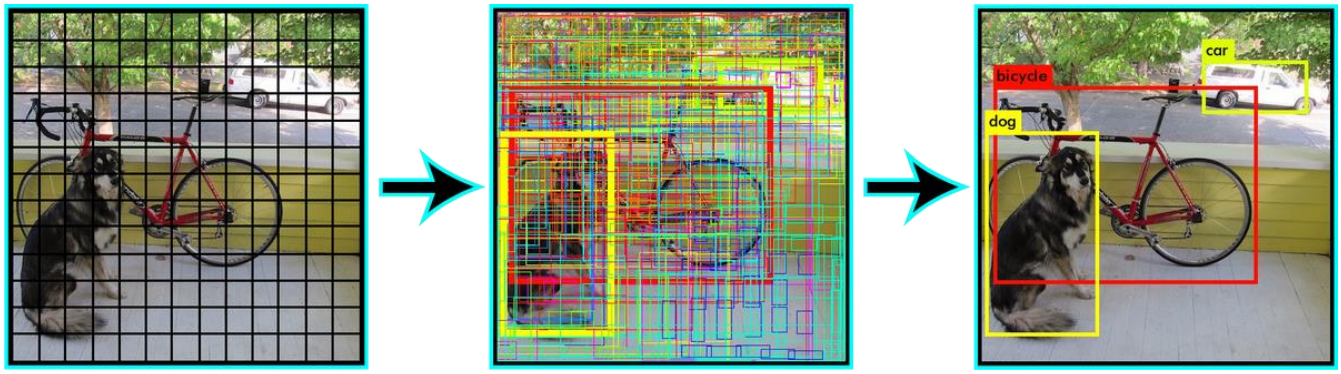


Рисунок 1.3 – Приклад роботи YOLO [5]

YOLO – це один з найбільш ефективних алгоритмів для розпізнавання об'єктів на зображеннях та відео. Його основною перевагою є висока швидкість і точність, завдяки яким він здатний здійснювати розпізнавання в реальному часі. Особливість цього підходу полягає в тому, що всі об'єкти на зображенні обробляються за один прохід через нейронну мережу, що значно прискорює процес. Це відрізняє YOLO від традиційних методів, які спочатку генерують потенційні області для обробки, а потім працюють з ними окремо. Завдяки цьому YOLO здатний виявляти об'єкти одночасно і на великій кількості зображень.

YOLO розбиває зображення на сітку і для кожної клітинки передбачає ймовірність наявності об'єкта, його координати, а також ймовірний клас об'єкта. Після цього, використовуючи алгоритм Non-Maximum Suppression (NMS), модель відкидає дубльовані прогнози і залишає тільки найбільш впевнені об'єкти. Це дозволяє алгоритму ефективно і швидко виявляти об'єкти навіть при перекритті або малих розмірах.

Незважаючи на численні переваги, YOLO має й певні обмеження. Він погано справляється з дуже маленькими об'єктами або з об'єктами, що знаходяться в

складних умовах, наприклад, у низькому освітленні. Для досягнення кращих результатів також потрібне потужне апаратне забезпечення, хоча для менш вимогливих задач існують легші версії алгоритму [5].

.

1.2 Застосування нейромережі в розпізнаванні образів

1.2.1 **Визначення нейронної мережі**

Нейронні мережі – це набір алгоритмів, натхненних структурою та функціонуванням мозку людини, призначених для розпізнавання шаблонів. Вони обробляють сенсорні дані, застосовуючи специфічні методи машинного сприйняття, маркування або класифікації вихідних результатів. Шаблони, які нейронні мережі здатні виявляти, є числовими та представлені у вигляді векторів. Усі дані реального світу, такі як зображення, звук, текст або часові ряди, мають бути переведені у відповідні векторні форми для подальшої обробки. На рис. 1.3 зображено схему нейрона.

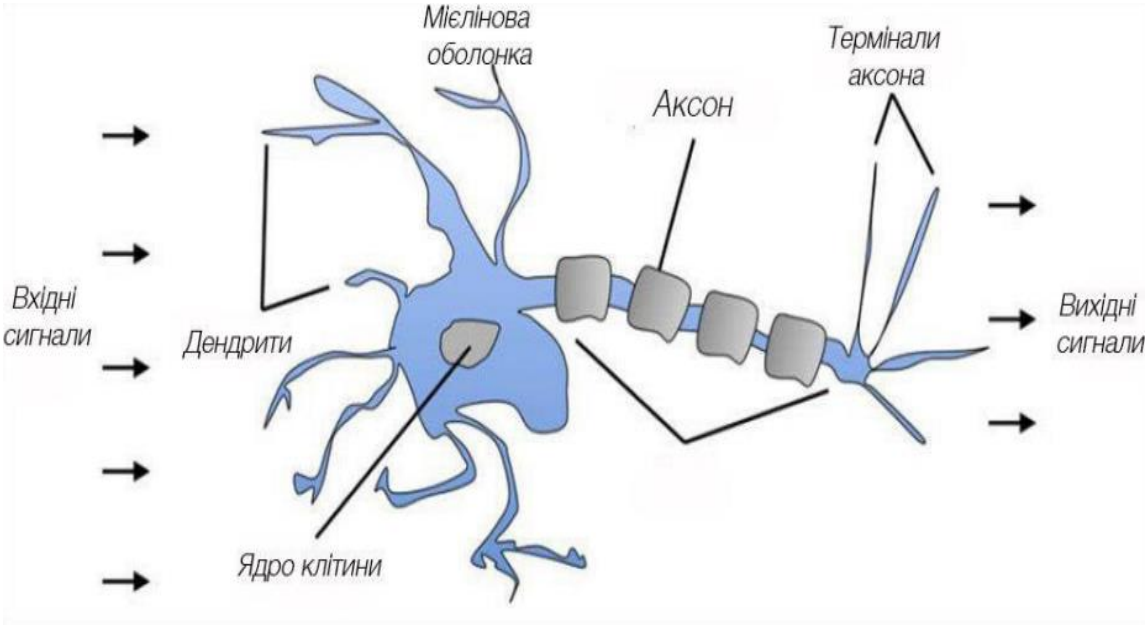


Рисунок. 1.3 – Зображення нейрона [6]

Нейронні мережі використовуються для кластеризації та класифікації даних, діючи як додатковий рівень обробки, що допомагає упорядкувати та структурувати

інформацію, яку ви зберігаєте та аналізуєте. Вони дозволяють групувати немарковані дані за подібністю між прикладними входами та класифікувати їх, якщо доступний мічений набір даних для навчання. Крім того, нейронні мережі можуть витягувати ознаки, які потім використовуються іншими алгоритмами кластеризації та класифікації. Тому глибокі нейронні мережі можна розглядати як ключові компоненти більш комплексних додатків машинного навчання, що включають алгоритми для навчання з підкріпленням, класифікації та регресії.

Класифікаційні завдання в машинному навчанні базуються на використанні мічених наборів даних. Це означає, що людина повинна передати свої знання у вигляді міток, які описують відповідність між вхідними даними та очікуваними результатами. Такий підхід називається наглядовим навчанням. Він широко застосовується для вирішення різноманітних задач, серед яких:

– ідентифікація осіб на зображеннях або відео, аналіз міміки для визначення емоцій (наприклад, гнів, радість);

– виявлення та класифікація об'єктів на зображеннях, таких як дорожні знаки, пішоходи або маркери смуг руху;

– розпізнавання жестів та рухів на відеозаписах, що може бути корисним для інтерактивних систем або управління пристроями;

– ідентифікація мовців, трансформація голосових повідомлень у текст, розпізнавання емоційного стану за допомогою аналізу тональності та інтонації;

– визначення спам-листів у електронній пошті, виявлення шахрайських повідомлень у страхових заявках або інших документах.

Будь-які мітки, створені людиною, а також будь-які результати, що мають взаємозв’язок із вхідними даними, можуть бути використані для навчання нейронної мережі. Завдяки класифікації, глибоке навчання дозволяє встановлювати відповідність між, наприклад, пікселями зображення та іменем людини. Такий підхід можна вважати статичним прогнозом, оскільки він визначає фіксовані зв’язки між даними та мітками.

Однак можливості глибокого навчання не обмежуються лише статичними завданнями. З достатньою кількістю релевантних даних система може навчитися прогнозувати майбутні події, виходячи з поточних. У такому разі модель встановлює регресивну залежність між минулим і майбутнім. Це означає, що майбутня подія стає свого роду маркером, який використовується для навчання нейронної мережі і прогнозування результатів у нових умовах.

Іншим важливим завданням нейронних мереж є кластеризація, або групування, що полягає у виявленні подібності між даними. На відміну від класифікації, кластеризація не потребує наявності міток. Цей тип навчання називається непідконтрольним навчанням, і він працює з немаркованими даними, які складають більшість інформації у світі.

Один із ключових принципів машинного навчання говорить: чим більше даних використовується для навчання алгоритму, тим точнішим буде результат. Завдяки цьому непідконтрольне навчання має значний потенціал для створення високоточних моделей, адже воно може використовувати великі обсяги необроблених даних. Основні напрямки застосування кластеризації є знаходження та порівняння схожих документів, зображень чи аудіофайлів. Наприклад, пошук зображень із подібними рисами або аудіотреків із схожою тональністю. Та виявлення аномалій, відхилень чи незвичної поведінки в даних. Це особливо корисно для завдань, де аномалії пов’язані з небажаними подіями, такими як шахрайство, кіберзагрози чи технічні збої [7].

1.2.2 **Елементи нейронної мережі**

Глибоке навчання - це термін, що використовується для опису багатошарових нейронних мереж, які складаються з кількох послідовних шарів. Кожен шар мережі складається з вузлів.

Вузол - це елемент, у якому виконуються обчислення, натхненні принципом роботи біологічного нейрона в людському мозку. У природному нейроні активація відбувається, коли він отримує достатню кількість сигналів. Аналогічно, у

штучному вузлі вхідні дані комбінуються з вагами - числовими коефіцієнтами, які модифікують значення входів. Ваги визначають, наскільки важливим є кожен сигнал для виконання завдання, яке вивчає алгоритм. Наприклад, вони допомагають виявити, які вхідні дані є найкориснішими для класифікації.

Результати цих комбінацій (добутки входів на відповідні ваги) підсумовуються. Сума проходить через функцію активації, яка вирішує, чи повинен вузол активуватися (передавати сигнал далі) і наскільки сильним буде цей сигнал. Якщо сигнал передається далі, це означає, що нейрон активований, і вплив цього сигналу поширюється на наступні вузли в мережі, формуючи остаточний результат.

Цей процес є ключовим для глибоких нейронних мереж, оскільки багатошаровість дозволяє моделі автоматично виділяти складні та багаторівневі залежності у вхідних даних. На рис. 1.4 представлено схематичне зображення нейрона, яке ілюструє цей процес: вхідні дані, ваги, підсумовування, функцію активації та результат [8].

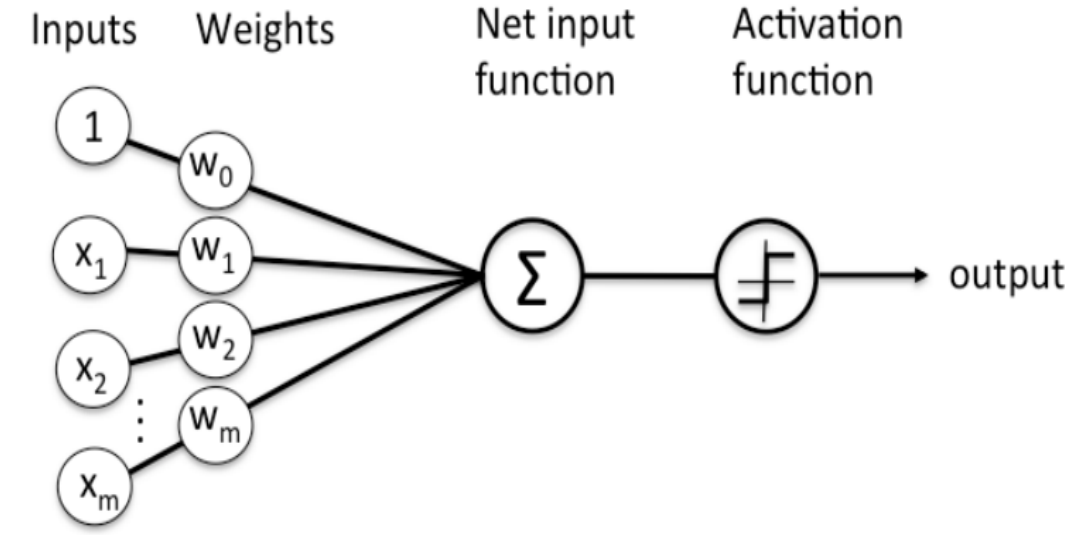


Рисунок 1.4 – Схема зображення одного вузла [9]

Шар вузлів - це сукупність нейронних елементів (вузлів), які виконують обчислення на одному рівні нейронної мережі. Ці вузли працюють як своєрідні перемикачі, які активуються (вмикаються) або залишаються неактивними (вимикаються) залежно від оброблюваних вхідних даних.

Ключова особливість шарів у нейронній мережі полягає в тому, що вихід кожного шару слугує вхідними даними для наступного шару. Цей ланцюжок починається з вхідного шару, який приймає початкові дані (наприклад, зображення, текст чи числові значення). На рис 1.5 зображено приклад простої нейромережі

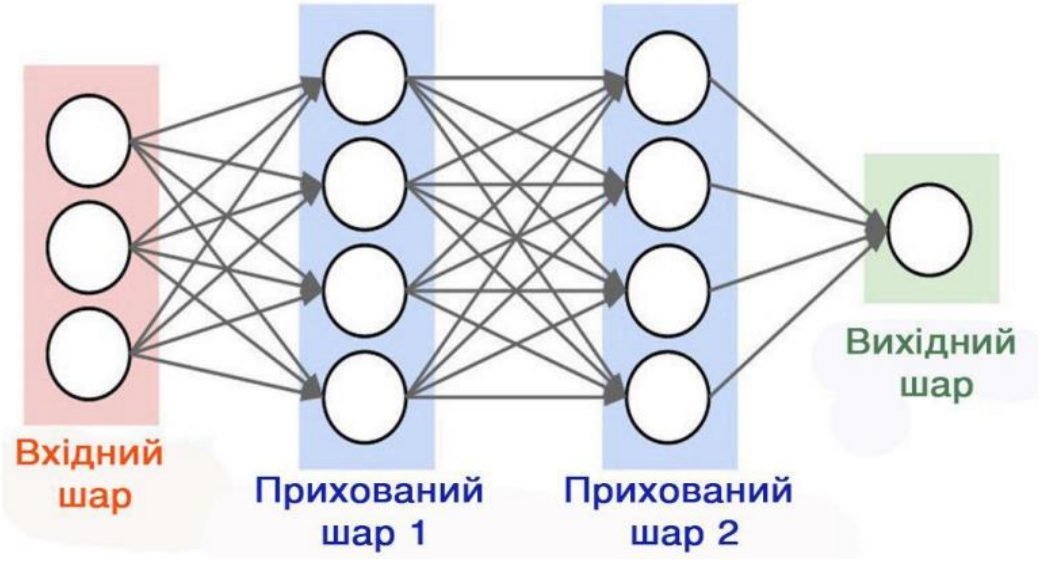


Рисунок 1.5 – Приклад простої нейромережі [10]

Сполучення регульованих ваг моделі з функціями введення полягає в тому, як ми визначаємо значення цих характеристик та їх вплив на класифікацію та кластеризацію нейронної мережі. Ваги служать для коригування значущості кожної ознаки в процесі навчання, дозволяючи мережі оптимізувати своє сприйняття входу та покращити здатність до класифікації чи групування даних. Правильне налаштування ваг і функцій введення забезпечує точність і ефективність нейронної мережі при розпізнаванні шаблонів або класифікації інформації [11].

1.2.3 **Основні поняття про глибокі нейронні мережі**

Мережі глибокого навчання вирізняються своєю глибиною – тобто кількістю шарів вузлів, які дані проходять у багатоступеневому процесі розпізнавання шаблонів. На відміну від ранніх нейронних мереж, що зазвичай називалися неглибокими, такі мережі складалися з одного вхідного шару, одного вихідного шару та одного або кількох прихованих шарів між ними. Вважається, що мережа є глибокою, якщо має більше трьох шарів (включно з вхідним і вихідним). У цьому

випадку термін "глибоке навчання" означає наявність кількох прихованих шарів, що дозволяють мережі моделювати складні залежності та абстракції.

Глибоке навчання, також відоме як глибоке структуроване навчання, ієрархічне навчання або глибоке машинне навчання, - це підгалузь машинного навчання, яка базується на використанні багатошарових алгоритмів. Ці алгоритми здатні будувати моделі високорівневих абстракцій даних.

У найпростішому випадку нейронна мережа може мати лише два набори нейронів вхідний шар, який приймає початкові дані і вихідний шар, який передає кінцевий результат.

Однак у глибокій мережі між вхідним і вихідним шарами існує багато прихованих шарів, кожен з яких виконує свої трансформації даних. Кожен прихований шар може складатися з лінійних і нелінійних перетворень, що дозволяє поступово виділяти все більш складні та багаторівневі особливості даних.

У мережах із глибоким навчанням кожен шар вузлів налаштовується для вивчення специфічних функцій на основі вихідних результатів попереднього шару. Цей підхід дозволяє моделі поступово будувати все складніші уявлення про вхідні дані, оскільки кожен наступний шар агрегує та рекомбінує особливості, вивчені на попередньому рівні

Мережі з глибоким навчанням мають здатність працювати з величезними наборами даних, які містять мільярди параметрів. Вони використовують нелінійні функції активації, щоб моделювати складні взаємозв’язки між вхідними даними. Завдяки багатошаровій структурі та нелінійним перетворенням, такі мережі можуть вирішувати завдання, які раніше вважалися занадто складними для класичних алгоритмів машинного навчання.

На рис. 1.6 схематично показано, як мережа вивчає особливості з поступово зростаючою складністю та абстракцією. Наприклад, у задачах комп’ютерного зору:

Перший шар може виявляти прості геометричні патерни. Наступні шари – поєднання цих патернів у більш складні структури, такі як обличчя, автомобілі чи

літери. Завершальні шари можуть розпізнавати цілі сцени або інтерпретувати контекст.

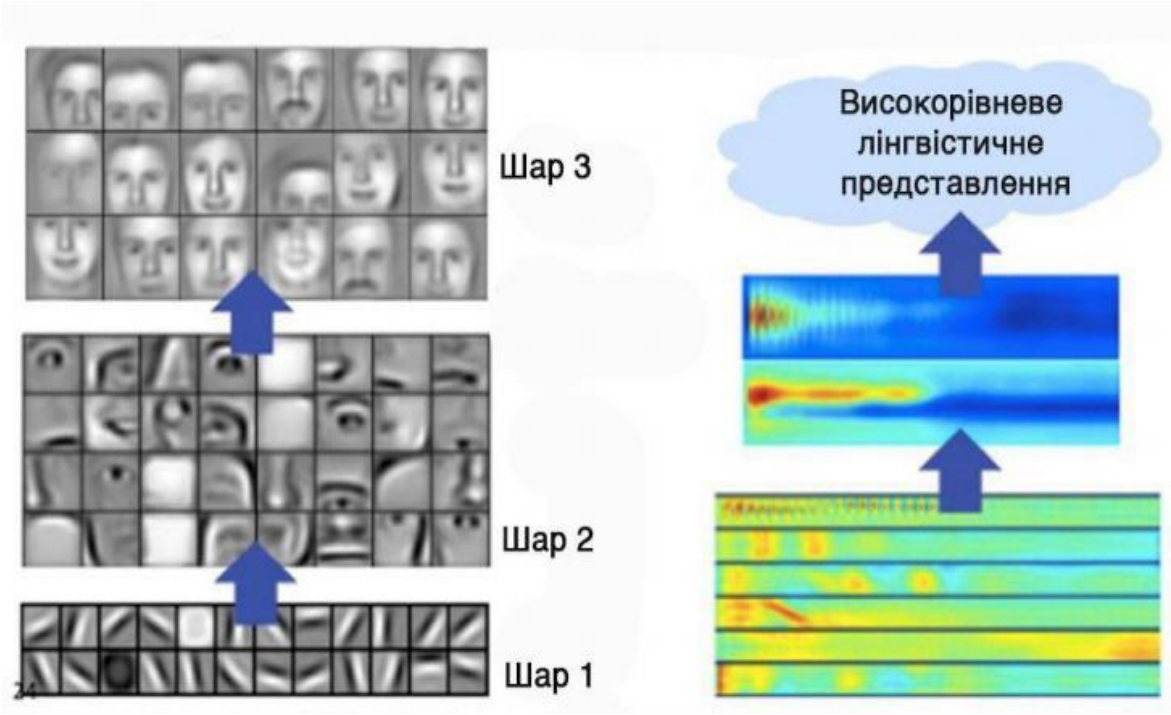


Рисунок 1.6 – Ієрархія особливостей [12]

Глибоке навчання має унікальну здатність виявляти приховану структуру в неструктурованих даних, які складають більшість інформації у світі. Неструктуровані дані, такі як зображення, тексти, відео та аудіо записи, не мають чіткої організації або попередньої мітки, що робить їх складними для традиційних алгоритмів. Однак глибоке навчання дозволяє ефективно обробляти ці дані, автоматично виявляючи подібності та класифікуючи їх без участі людини. Наприклад, система може взяти мільйони зображень і згрупувати їх за подібністю, визначаючи, які з них містять котів, а які – криголами. Це є основою технології "розумних фотоальбомів", де фотографії автоматично сортуються за виявленими особливостями.

З часом глибоке навчання також дозволяє групувати дані відповідно до нормальної та аномальної поведінки. Наприклад, у медицині, якщо дані з часового ряду збираються зі смарт-телефона, можна оцінити стан здоров'я або звички

користувача, а в промисловості – прогнозувати та запобігати поломкам техніки, аналізуючи дані з сенсорів.

Однією з ключових переваг глибокого навчання є здатність автоматично витягувати ознаки з даних без необхідності ручного втручання. Це суттєво відрізняє глибоке навчання від традиційних методів машинного навчання, де вилучення ознак часто потребує значних зусиль з боку експертів. Завдяки цьому глибоке навчання дозволяє швидше адаптувати моделі до нових даних і застосовувати їх до широкого кола завдань, не покладаючись на обмежене число фахівців. Таким чином, глибоке навчання робить можливим ефективне оброблення великих обсягів даних, надаючи потужні інструменти для аналізу та прогнозування в різних галузях.

Під час навчання глибокі нейронні мережі працюють таким чином, що кожен рівень вузлів автоматично вивчає функції, намагаючись відновити вхідні дані, з яких він черпає свої зразки. Цей процес має на меті мінімізувати різницю між передбаченнями мережі та розподілом ймовірностей самих вхідних даних. Іншими словами, мережа намагається реконструювати вхід, знижуючи помилки у своїх прогнозах.

У процесі навчання ці мережі навчаються розпізнавати кореляції між певними ознаками та оптимальними результатами. Вони створюють зв’язки між сигналами характеристик і тим, що ці функції представляють, незалежно від того, чи йдеться про повну реконструкцію або роботу з міченими даними. Це дозволяє мережам глибокого навчання точно виділяти важливі патерни та ознаки, необхідні для прийняття правильних рішень [13].

Навчання в таких мережах завершується на вихідному шарі, де розташовується класифікатор, що призначає ймовірність певному результату або мітці. Наприклад, якщо вхідними даними є зображення, мережа з глибоким навчанням може визначити, що ймовірність того, що на зображенні зображена людина, становить 90 відсотків. Таким чином, в кінцевому результаті ми

отримуємо прогнозовані дані, що дозволяють точно класифікувати або зробити висновок щодо предмета чи явища на основі обробленої інформації.

1.3 Вимоги до Системи

Система повинна відповідати наступним загальним вимогам:

– Виявлення захворювань рослин. Система повинна автоматично розпізнавати ознаки захворювань рослин за допомогою аналізу зображень, що отримуються з камер відеоспостереження. Використання нейронної мережі дозволяє ідентифікувати різні типи захворювань на ранніх стадіях розвитку.

– Обробка зображень у реальному часі. Система повинна забезпечувати обробку зображень у реальному часі для своєчасного виявлення змін у стані рослин. Підтримка потокового відео дозволяє безперервно контролювати стан усіх рослин у теплиці.

– Оповіщення користувачів. У разі виявлення ознак хвороби або будь-яких відхилень у стані рослин, система повинна автоматично надсилати сповіщення відповідним користувачам через мобільний додаток або інші канали комунікації (e-mail, SMS).

– Підтримка розподіленої архітектури. Система повинна мати можливість працювати в умовах розподіленої архітектури з декількома вузлами, що відповідають за моніторинг різних ділянок теплиці. Обробка даних здійснюється як на місцевому рівні (вузлові пристрої), так і на центральному сервері для забезпечення безперебійного функціонування системи.

Розроблене апаратно-програмне забезпечення повинно працювати у наступних режимах:

Режим моніторингу: Безперервне отримання зображень з камер відеоспостереження, їх попередня обробка та аналіз за допомогою нейронної мережі для виявлення ознак захворювань.

Режим сповіщення: У разі виявлення захворювання система повинна автоматично надсилати сповіщення користувачам через мобільний додаток або e-mail з детальною інформацією про виявлену проблему.

Режим обробки запитів: Користувачі мають можливість отримувати звіти про стан рослин у теплиці за будь-який період, а також переглядати статистику виявлених захворювань.

Ці вимоги закладають основу для створення сучасної системи, яка інтегрує технології нейронних мереж, комп'ютерного зору та розподілених обчислень, що дозволить підвищити ефективність аграрного виробництва та зменшити втрати врожаю.

Висновки до розділу 1

У цьому розділі було здійснено аналіз сучасних програмних комплексів для розпізнавання образів, таких як FineReader та YOLO. Окремо описано різні підходи до розпізнавання образів, надано визначення нейронних та глибоких нейронних мереж, а також розглянуті основні елементи нейронної мережі.

Було виділено дві основні групи методів розпізнавання образів: перша включає методи, що використовують класифіковану навчальну матрицю для формування вирішальних правил, а друга – методи автоматичної класифікації, які здатні обробляти некласифіковані дані. Також було проаналізовано критерії якості розпізнавання образів та визначено специфікації вимог до системи.

Було сформульовано специфікацію вимог до апаратно-програмного забезпечення системи health-моніторингу теплиць, що дозволить забезпечити автоматизацію процесів моніторингу стану рослин. Визначені функціональні вимоги охоплюють ключові аспекти роботи системи: автоматичне виявлення хвороб, обробку зображень у реальному часі, оповіщення користувачів та збереження даних.

2 ПРОЄКТУВАННЯ АПАРАТНО-ПРОГРАМНОГО МОДУЛЯ ТА ВИБІР СКЛАДУ ТЕХНІЧНИХ І ПРОГРАМНИХ ЗАСОБІВ

2.1 Проєктування апаратного забезпечення

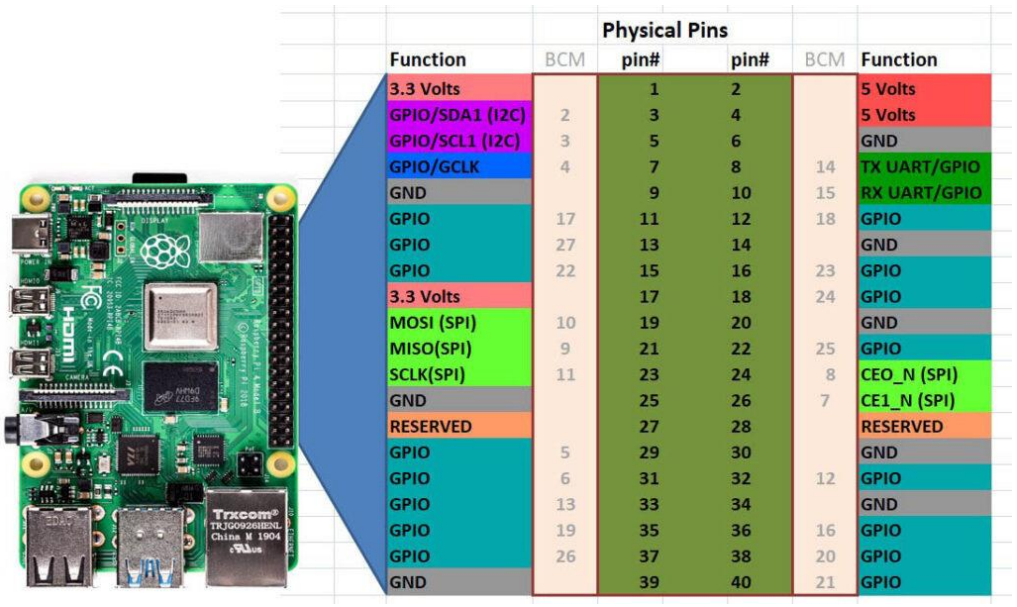
Розробка апаратного забезпечення для системи health-моніторингу теплиць вимагає ретельного підбору компонентів та їхньої інтеграції для забезпечення злагодженої роботи всієї системи. Враховуючи специфіку завдання, критичне значення мають такі аспекти, як достатня обчислювальна потужність для роботи з алгоритмами штучного інтелекту, енергоефективність, компактність, а також надійність і можливість масштабування системи. Для цієї мети ідеально підходять одноплатні комп’ютери, що поєднують високу продуктивність та гнучкість в роботі з різними сенсорами та камерами.

Одноплатні комп’ютери, забезпечують достатню потужність для виконання завдань комп’ютерного зору та роботи нейронних мереж у реальному часі. Вони підтримують роботу з бібліотеками для обробки зображень (OpenCV, TensorFlow) та мають інтерфейси для підключення периферійного обладнання через **GPIO** (general-purpose input/output).

Більшість сучасних одноплатних комп’ютерів оснащені універсальними інтерфейсами GPIO, які дозволяють підключати та керувати периферійними пристроями, такими як датчики, світлодіоди, реле чи камери.

Інтерфейс GPIO забезпечує гнучке конфігурування контактів для роботи як у режимі введення (input), так і виведення (output), залежно від потреб проєкту. Це дозволяє організувати двосторонню передачу сигналів між контролером та підключеними пристроями [14].

Окрім керуючих контактів, GPIO включає окремі спеціалізовані виводи для забезпечення живлення периферійних компонентів (зазвичай 3.3V, 5V або GND). Ці піни слугують виключно для живлення та не можуть бути використані для передачі даних. На Рисунку 2.1 зображено інтерфейс GPIO для Raspberry Pi 4, що демонструє його конфігурацію та розташування виводів.



		Physical Pins			
Function	BCM	pin#	pin#	BCM	Function
3.3 Volts		1	2		5 Volts
GPIO/SDA1 (I2C)	2	3	4		5 Volts
GPIO/SCL1 (I2C)	3	5	6		GND
GPIO/GCLK	4	7	8	14	TX UART/GPIO
GND		9	10	15	RX UART/GPIO
GPIO	17	11	12	18	GPIO
GPIO	27	13	14		GND
GPIO	22	15	16	23	GPIO
3.3 Volts		17	18	24	GPIO
MOSI (SPI)	10	19	20		GND
MISO(SPI)	9	21	22	25	GPIO
SCLK(SPI)	11	23	24	8	CEO_N (SPI)
GND		25	26	7	CE1_N (SPI)
RESERVED		27	28		RESERVED
GPIO	5	29	30		GND
GPIO	6	31	32	12	GPIO
GPIO	13	33	34		GND
GPIO	19	35	36	16	GPIO
GPIO	26	37	38	20	GPIO
GND		39	40	21	GPIO

Рисунок 2.1 – Інтерфейс GPIO Raspberry Pi 3b

Процес підключення компонентів до одноплатного комп’ютера вимагає ретельного планування та перевірки коректності з’єднань. Це можна зробити за допомогою спеціалізованих програм, таких як Fritzing. Цей інструмент дозволяє створювати принципові електричні схеми, моделювати роботу компонентів та тестувати їхню взаємодію до фізичної реалізації.

Завдяки простому та інтуїтивному інтерфейсу, Fritzing широко використовується для прототипування апаратних систем на базі плат Arduino, однак він також підтримує роботу з різними одноплатними комп’ютерами, включаючи Raspberry Pi та NVIDIA Jetson.

Моделювання принципової схеми в Fritzing дає змогу відобразити зв'язки між усіма компонентами системи та уникнути можливих помилок при їх підключенні. Принципові електричні схеми показують умовні позначення компонентів та взаємозв’язки між ними, що дозволяє детально спланувати структуру системи. На рис. 2.2 наведено приклад принципової схеми, створеної в середовищі Fritzing, яка демонструє послідовність підключення сенсорів до контролера.

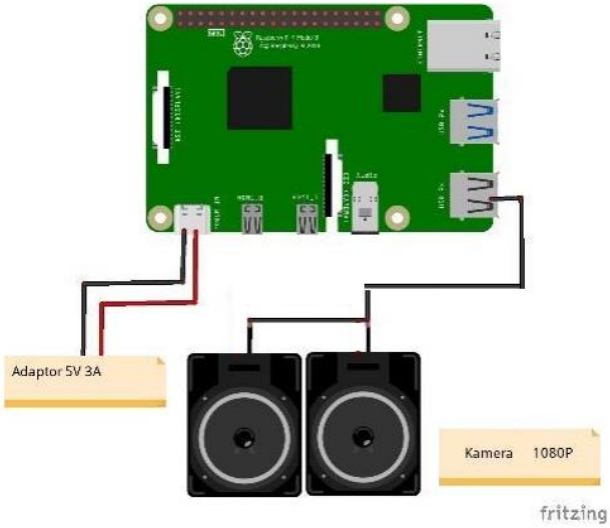


Рисунок 2.2 – Відображення принципової схеми засобами Fritzing

Використання таких програмних інструментів для попереднього моделювання є критично важливим для забезпечення надійної роботи розроблюваного програмно-апаратного комплексу. Це дозволяє ще на етапі проєктування проаналізувати можливі помилки, внести коригування та підвищити ефективність інтеграції всіх компонентів у єдину систему.

2.2 Проєктування програмного забезпечення

Розробка програмного забезпечення для програмно-апаратних комплексів – це складний і багатоетапний процес, який потребує ретельного планування. Перед початком розробки необхідно визначити структуру та алгоритми роботи системи.

Для опису структури програмного забезпечення часто використовується уніфікована мова моделювання (Unified Modeling Language, UML). Це графічний інструмент, що дозволяє створювати моделі компонентів ПЗ на основі принципів об'єктно-орієнтованого програмування. UML також може застосовуватися для візуалізації бізнес-процесів, які тісно пов'язані з функціонуванням програмного забезпечення.

UML включає різні типи діаграм, зокрема діаграми станів, класів, прецедентів та інші. Їхнє комплексне використання дозволяє розробникам отримати цілісне уявлення про архітектуру ПЗ, забезпечуючи всебічне розуміння

системи. Для повного опису архітектури проєктованого програмного забезпечення рекомендується використовувати набір UML-діаграм відповідно до етапів розробки.

Одним із найбільш поширених інструментів при проєктуванні складних програмних систем є **діаграма класів (Class Diagram)**. Це статична структурна діаграма, яка відображає:

- **класи**, що визначають структуру системи;
- **методи** та **атрибути** класів;
- **взаємозв’язки** між класами, включаючи асоціації, агрегації та композиції.

Існує кілька варіацій діаграм класів, що використовуються на різних етапах розробки:

- **концептуальна діаграма класів:** відображає лише приклади класів, що належать до певної предметної області, допомагаючи сформувати загальне бачення структури;
- **діаграма класів на етапі специфікації:** створюється під час формування вимог до ПЗ і є невід'ємною частиною процесу проєктування;
- **діаграма класів етапу реалізації:** відображає реальні елементи системи, які будуть присутні в коді. Цей тип діаграми забезпечує відповідність між проєктною документацією та реалізованим програмним продуктом.

Таким чином, використання UML надає можливість побудувати чітку та зрозумілу модель програмного забезпечення, що полегшує комунікацію між розробниками та іншими учасниками проєкту, а також сприяє успішній реалізації поставлених завдань.

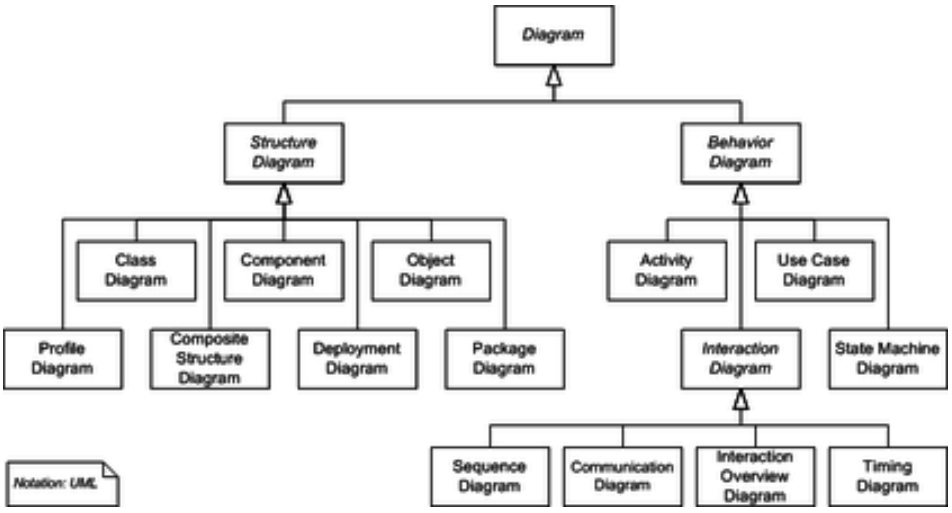


Рисунок 2.3 – Ієрархія діаграм UML

Поділ типів діаграм класів, описаний раніше, не впливає на інструменти, які використовуються для їх створення. Програмне забезпечення, призначене для моделювання, відноситься до категорії комп'ютерних засобів автоматизації розробки програмного забезпечення (CASE). Інструменти CASE не лише дозволяють створювати різні моделі, але й можуть частково генерувати код на основі розроблених моделей. Список програм, що належать до цього класу, досить великий, проте слід виділити **SoftwareIdeasModeler**.

SoftwareIdeasModeler надає можливість створення моделей UML, ERD-діаграм, блок-схем та інших візуальних представлень. Створені моделі можна експортувати у вигляді графічних зображень для подальшого використання у документації або ж у форматі програмного коду для мов програмування, таких як C++, C#, PHP, Python та інші [15].

2.3 Застосування моделі нейронної мережі для розпізнавання зображень

Штучні нейронні мережі (ШНМ; англ. Artificial Neural Networks, ANN) являють собою обчислювальні системи, які базуються на принципах роботи біологічних нейронних мереж, що формують мозок тварин. Ці мережі здатні самостійно навчатися та вдосконалювати свої результати, використовуючи дані для аналізу та виявлення закономірностей без потреби в спеціальному програмуванні під конкретні задачі.

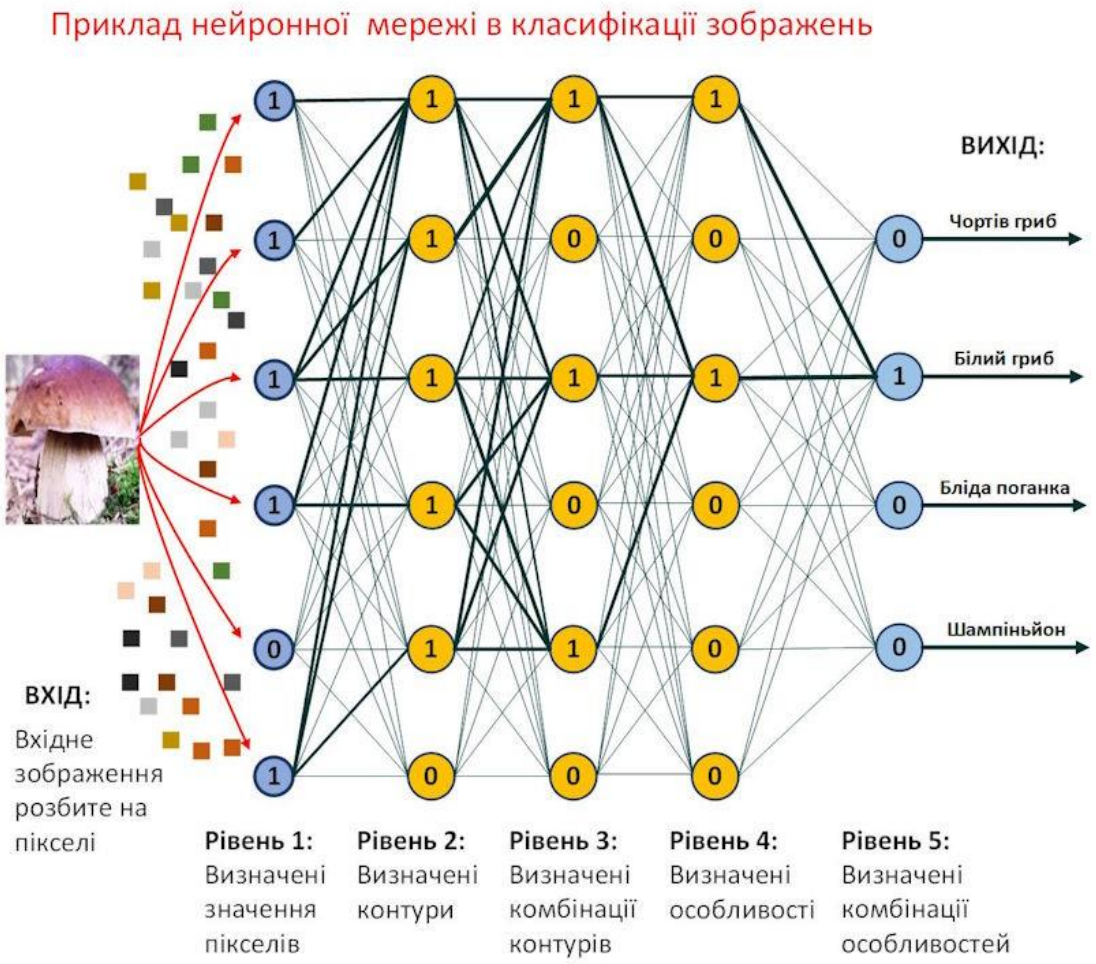


Рисунок 2.4 – Схематичне відображення моделі нейронної мережі

Мова програмування Python пропонує кілька потужних фреймворків для роботи зі штучним інтелектом, таких як OpenCV, TensorFlow та PyTorch та інші. Ці фреймворки забезпечують необхідні інструменти для створення, навчання та застосування моделей нейронних мереж, що робить їх надзвичайно зручними для розробки систем розпізнавання мовлення та інтеграції в різні побутові пристрої [16]. Для подальшої роботи розглянемо 2 фреймворки такі як OpenCV та TensorFlow

OpenCV (Open Source Computer Vision Library) – це одна з популярніших бібліотека для комп'ютерного зору з відкритим кодом, яка була створена компанією Intel у 1999 році. Вона використовуваних бібліотек для обробки зображень і відео в реальному часі, підтримує різні мови програмування,

включаючи Python, C++, Java, MATLAB, і може працювати на багатьох платформах, таких як Windows, Linux, macOS, Android та інші.

OpenCV дозволяє виконувати безліч складних завдань комп'ютерного зору, включаючи розпізнавання обличчя, детекцію об'єктів, сегментацію зображень, аналіз відео, а також розробку систем для автономних транспортних засобів, розпізнавання тексту та медичні дослідження. У бібліотеці реалізовані алгоритми для детекції облич, зокрема методи Fisherfaces, гістограми локальних бінарних шаблонів та власні обличчя. OpenCV також пропонує функції для обробки та аналізу зображень у реальному часі, а також підтримує багато інтерфейсів, що дозволяють інтегрувати його в різні програмні середовища [17].

Основні можливості технології комп'ютерного зору охоплюють кілька ключових напрямків. Одним із таких напрямків є технологія **виявлення об'єктів**, яка застосовується для розпізнавання зображень і визначення місцезнаходження конкретних об'єктів на відео чи зображеннях, таких як автомобілі, люди, тварини, а також окремі частини чи обладнання в промисловому виробництві. Виявлення кількох об'єктів за допомогою машинного навчання реалізовано за допомогою бібліотеки OpenCV в рамках платформи Viso Suite, що дає змогу ефективно виконувати пошук і класифікацію різних об'єктів у реальному часі.

Сегментація зображень застосовує алгоритми обробки зображень для поділу зображення на окремі сегменти або області. Це дозволяє спростити, змінити чи покращити зображення, часто для подальшої обробки або виконання інших завдань комп’ютерного зору. Сегментація є важливим етапом у завданнях, таких як автономне водіння, де для визначення меж дороги та об'єктів на ній застосовуються методи сегментації зображень.

Розпізнавання поз і жестів людини є важливою частиною багатьох сучасних систем, що дозволяють інтерпретувати та розуміти рухи і жести людей за допомогою відеоаналітики. Це включає розпізнавання рухів тіла, рук або обличчя, а також аналіз пози людини на основі ключових точок, таких як суглоби і кінцівки. Технології аналізу рухів, зокрема визначення пози об’єкта в 3D-просторі,

використовуються для інтерактивних додатків, таких як ігри чи системи безпеки. Прикладом реалізації таких технологій є аналіз рухів за допомогою оцінки пози за допомогою ключових точок на платформі Viso Suite.

Автоматичне розпізнавання облич є важливою технологією для ідентифікації людей, що здійснюється шляхом виявлення людського обличчя на зображенні та зіставлення його з базою даних за допомогою виявлених ознак, таких як риси обличчя. OpenCV надає інструменти для реалізації розпізнавання облич у реальних програмах, що має застосування в системах безпеки, мобільних додатках, маркетингу та багатьох інших сферах.

Доповнена реальність (AR augmented reality) дозволяє інтегрувати віртуальні об'єкти в реальний світ у реальному часі, що відкриває нові можливості для взаємодії з фізичним середовищем. AR технології застосовуються для створення інтерактивних додатків, які поєднують фізичний і віртуальний світи, наприклад, у навчанні, розвагах, архітектурному проектуванні чи ремонті. Завдяки комп’ютерному зору, AR здатна доповнювати фізичний світ перцептивною інформацією, створеною за допомогою комп’ютерних алгоритмів [18].

Ці технології відкривають безліч можливостей для інновацій у різних сферах, від промисловості та безпеки до медичних і розважальних застосувань. Існує багато різноманітних задач які чудово розв’язуються за допомогою OpenCV, деякі з них наведені нижче

- розпізнавання обличчя;
- автоматизований огляд і спостереження;
- інтерактивні арт-інсталяції;
- підрахунок транспортних засобів на магістралях разом із визначенням їх швидкості;
- розпізнавання об'єктів;
- виявлення аномалій (дефектів) у виробничому процесі, таких як браковані вироби;
- створення 3D-структур з руху в кінематографії;

– навігація та керування автомобілем без участі водія;
– аналіз медичних зображень;
– розпізнавання реклами на телеканалах.

Крім того, OpenCV активно використовує алгоритми машинного навчання для виконання складних завдань, таких як класифікація та кластерифікація, а також інтеграцію з платформами на базі GPU, що дозволяє значно прискорити обробку даних (за допомогою CUDA та OpenCL). Завдяки відкритому коду, OpenCV широко використовується як в академічних дослідженнях (MIT, Stanford, CMU, INRIA), так і в промисловості – такими компаніями, як Google, Facebook, Microsoft, Sony, Toyota, Siemens і IBM.

Одним із важливих аспектів OpenCV є її адаптивність до змінних умов. Оскільки комп'ютерне бачення часто працює з даними, які можуть бути викривлені через зміни освітлення, камери, рухи або механічні налаштування, OpenCV надає потужні інструменти для обробки та корекції таких спотворень (наприклад, усунення шуму або спотворень). Крім того, розробники можуть створювати складні конвеєри для обробки зображень, що об'єднують різні етапи обробки від збору даних до застосування алгоритмів для вирішення специфічних завдань комп'ютерного зору.

Сфери використання комп’ютерного зору величезні та постійно розширюються. Хоча більшість людей знайомі з його популярними застосуваннями, такими як системи безпеки, відеоспостереження чи безпілотні автомобілі, менше уваги приділяється його потенціалу в конкретних галузях. Наприклад, у промисловому виробництві комп’ютерний зір застосовується для автоматизації перевірки якості продукції, моніторингу виробничих процесів та виявлення дефектів на етапі виробництва. Враховуючи швидкий розвиток цієї технології, можна очікувати ще більше інноваційних рішень, які змінять різні сфери діяльності.

2.4 Технологія машинного навчання TensorFlow

TensorFlow – це відкритий фреймворк для машинного навчання, розроблений компанією Google. Він дозволяє створювати та навчати моделі штучного інтелекту, зокрема нейронні мережі, для різноманітних задач, таких як обробка зображень, обробка природної мови, аналіз даних та багато інших.

У листопаді 2015 року Google випустила TensorFlow як програмне забезпечення з відкритим кодом під ліцензією, зробивши його доступним для використання, модифікації та внесення змін будь-ким. Це було важливим кроком бо TensorFlow був результатом багатьох років роботи та інновацій у Google, і рішення зробити його відкритим кодом відображало серйозне зобов'язання компанії сприяти розвитку штучного інтелекту та машинного навчання в публічному просторі.

З самого початку TensorFlow проектувався як не просто внутрішній інструмент для дослідників Google, а як потужний інструмент для глобальної спільноти. У своєму першому технічному документі команда TensorFlow визначила кілька ключових цілей:

– бути бібліотекою програмного забезпечення з відкритим вихідним кодом для чисельних обчислень, зокрема для застосунків машинного навчання;

– бути дуже гнучкою і розширюваною для підтримки досліджень нових алгоритмів та ідей;

– бути кросплатформною і розгортатися на широкому спектрі пристроїв, від смартфонів до центрів обробки даних;

– забезпечувати стабільність і зворотну сумісність для виробничих сценаріїв використання;

– включати еталонні реалізації поширених алгоритмів машинного навчання, щоб полегшити початок роботи.

Однією з ключових можливостей TensorFlow є автоматичне диференціювання, яке дозволяє автоматично обчислювати градієнти функцій для оптимізації параметрів моделей. Це важливо для процесу навчання, оскільки

оптимізатори, використовують ці градієнти для корекції ваг моделі з метою мінімізації функції втрат. Завдяки цьому, TensorFlow забезпечує ефективний процес тренування моделей, що є основою для створення нейронних мереж і інших алгоритмів машинного навчання.

TensorFlow підтримує широкий спектр можливостей, від простих регресійних моделей до складних нейронних мереж, таких як згорткові та рекурентні мережі. Однією з основних особливостей TensorFlow є її здатність працювати як на центральних процесорах (CPU), так і на графічних процесорах (GPU), що дає можливість значно прискорити обчислення для великих і складних моделей. Завдяки використанню GPU, процеси тренування можуть відбуватися набагато швидше, що особливо важливо при роботі з великими наборами даних.

Окрім основної бібліотеки TensorFlow, одним із найважливіших аспектів цього проєкту є екосистема, яка розвивалася навколо нього. Від інструментів для розробки та візуалізації моделей до масштабованої інфраструктури для розгортання в реальних умовах. Дана екосистема відкрила нові можливості й зробила машинне навчання доступним для більшої кількості розробників, ніж будь-коли раніше [19].

Ось деякі з основних компонентів екосистеми TensorFlow:

- **TensorBoard** – набір інструментів для візуалізації, що допомагає зрозуміти, налагодити та оптимізувати моделі. Він дозволяє зручно аналізувати структуру моделі та процес її навчання;
- **TensorFlow.js** – JavaScript-версія TensorFlow, яка дозволяє запускати моделі машинного навчання безпосередньо в браузері або в Node.js. Це дозволяє легко інтегрувати машинне навчання в веб-додатки;
- **TensorFlow Lite** – легковажне рішення для розгортання моделей на мобільних та вбудованих пристроях з обмеженими обчислювальними ресурсами, таких як смартфони, пристрої IoT та одноплатні комп'ютери;

– **TensorFlow Extended** – платформа для розгортання ML-пайплайнів в реальних умовах, яка допомагає управляти всім життєвим циклом моделі – від навчання до обслуговування й моніторингу;

– **TensorFlow Datasets і TensorFlow Hub** – ресурси, які роблять доступ до якісних наборів даних та перероблених компонентів моделей надзвичайно простим, дозволяючи спростити процес розробки моделей;

– **Keras** – спочатку незалежний проєкт, ця високорівнева API для побудови нейронних мереж була інтегрована безпосередньо в TensorFlow і стала основною частиною бібліотеки, що робить її більш доступною для початківців.

Широта та зрілість цієї екосистеми стали ключовими факторами успіху та впливу TensorFlow [20].

2.5 Архітектура програмного забезпечення

Архітектура ПЗ побудована з використанням принципів розподіленої обробки та модульності. Це дозволяє розділити функціональні компоненти на окремі блоки, що взаємодіють через визначені інтерфейси. Основні компоненти системи включають:

– **Рівень збору та попередньої обробки даних**: Виконується на локальних вузлах (мікроконтролерах чи мікрокомп'ютерах), де збираються та обробляються зображення з камер;

– **Центральний сервер обробки та зберігання**: Збирає дані з усіх вузлів, зберігає їх у базі даних та проводить додаткову обробку, якщо локальні ресурси виявляються недостатніми;

– **Інтерфейси користувача:** Включають мобільний додаток або веб-інтерфейс для доступу до аналітичних даних та сповіщень.

Логіка розподіленої архітектури полягає у тому що, дані із сенсорів і камер збираються на локальних вузлах. Попередньо оброблені дані (зображення та ключові показники) передаються на мікроконтролер на якому проводиться додаткова обробка або підтвердження результатів локального аналізу. І у разі

виявлення проблем система надсилає сповіщення користувачам через мобільний додаток чи вебінтерфейс.

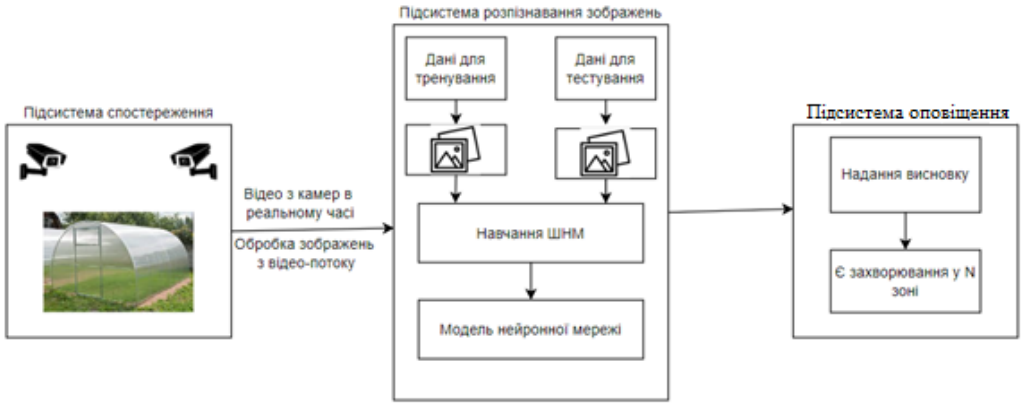


Рисунок 2.5 – Основні компоненти системи

Кожен модуль системи має конкретну функціональність, яка забезпечує взаємодію між компонентами та безперервну роботу системи.

Модуль спостереження

Модуль спостереження є основою системи health-моніторингу теплиць, відповідальною за збір даних про стан рослин та навколишнє середовище. Цей модуль включає в себе камери, сенсори та програмне забезпечення для обробки отриманої інформації.

Основні функції:

– збір даних: модуль здійснює автоматичний збір зображень рослин та параметрів середовища (температура, вологість, рівень освітленості) з використанням відповідних сенсорів і камер;

– регулярність збору: Збір даних здійснюється за заздалегідь заданим графіком (наприклад, кожні 30 хвилин) або у відповідь на певні події, такі як зміна температури чи вологості;

– форматування даних: Отримані дані з сенсорів та зображення проходять попередню обробку, у результаті чого формуються готові для аналізу набори даних.

Технології та обладнання

– камери: Використовуються високоякісні цифрові камери з можливістю зйомки в режимі низького освітлення, що дозволяє отримувати чіткі зображення рослин у будь-яких умовах;

– сенсори: Використовуються датчики для вимірювання температури, вологості, pH ґрунту та рівня освітленості, які підключені до системи збору даних;

– зв'язок: Модуль оснащений бездротовими модулями зв'язку, що забезпечує передачу даних на центральний сервер у реальному часі.

Модуль розпізнавання зображень

Модуль розпізнавання зображень відповідає за аналіз зібраних зображень рослин для виявлення захворювань і дефектів. Він використовує методи машинного навчання та комп'ютерного зору для здійснення класифікації стану рослин.

Основні функції

– обробка зображень: Зображення, отримані з модуля спостереження, проходять етапи обробки: усунення шуму, підвищення контрастності, нормалізація. Це дозволяє підготувати дані для подальшого аналізу;

– класифікація: Використовуючи нейронні мережі (зокрема, CNN), модуль класифікує зображення, визначаючи наявність захворювань або стресових станів рослин;

– збір статистики: Модуль веде облік результатів аналізу, фіксуючи типи виявлених захворювань, що дозволяє аналізувати їх поширення та розробляти рекомендації для користувачів.

Технології та методи

– машинне навчання: Модуль використовує попередньо навчені моделі нейронних мереж, які адаптуються до специфіки даних. Для тренування моделей використовуються великі набори даних, що включають як здорові рослини, так і ті, що мають різні захворювання;

– обробка зображень: Бібліотеки, такі як OpenCV і TensorFlow, забезпечують необхідні функції для обробки та аналізу зображень;

– алгоритми: Використовуються різні алгоритми, такі як градієнтний спуск для оптимізації нейронних мереж, а також методи аугментації даних для підвищення точності класифікації.

Модуль сповіщення

Модуль сповіщення відповідає за передачу інформації користувачеві про стан рослин та виявлені проблеми. Це критично важливий компонент системи, оскільки своєчасне сповіщення може суттєво вплинути на ефективність управління теплицями.

Основні функції

– оповіщення про стан: Модуль формує сповіщення для користувачів на основі результатів аналізу, зокрема про виявлені захворювання, необхідність проведення обробки рослин або зміни в умовах вирощування;

– налаштування сповіщень: Користувачі можуть налаштувати отримання сповіщень на різних рівнях: критичні (необхідно терміново вжити заходів) та інформаційні (додаткова інформація про стан рослин);

– канали сповіщення: Модуль може використовувати різні канали для сповіщення: мобільні застосунки, електронну пошту, SMS-повідомлення тощо.

Модулі спостереження, розпізнавання зображень та сповіщення є ключовими компонентами апаратно-програмного модуля health-моніторингу теплиць. Кожен з модулів виконує свої специфічні функції, які взаємодіють між собою, забезпечуючи цілісну і ефективну систему для моніторингу стану рослин та їх своєчасного догляду. Ця інтеграція дозволяє не тільки підвищити продуктивність теплиць, а й знизити ризики, пов'язані із захворюваннями рослин.

Висновки до розділу 2

В даному розділі було розглянуто проектування апаратно-програмного модуля для системи health-моніторингу теплиць. писано інтерфейс для підключення периферійного обладнання через GPIO.

Досліджено інструменти для моделювання апаратного забезпечення, зокрема, використання принципових схем. Розглянуто програмне забезпечення, яке підтримує цей процес проєктування.

Також надано опис інструментів для розробки програмного забезпечення, зокрема засобів для опису архітектури програмних систем. Вивчено основні моделі мови UML та їхнє застосування при створенні програмного забезпечення.

Розглянуто основні функції та особливості роботи сучасних бібліотеки для розпізнавання зображень за допомогою комп’ютерного зору OpenCV та бібліотеки TensorFlow для розробки та тренування моделей нейронних мереж

Було розроблено архітектуру апаратно-програмного забезпечення системи health-моніторингу теплиць, що дозволить забезпечити автоматизацію процесів моніторингу стану рослин. Розроблені модулі охоплюють ключові аспекти роботи системи: автоматичне виявлення хвороб, обробку зображень у реальному часі, оповіщення користувачів та збереження даних.

3 АПАРАТНО-ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Блок-схема процесів розподіленої системи моніторингу

Розподілена система моніторингу отримує дані з різних камер відео нагляду і оброблює їх для отримання потрібної нам інформації про хвороби рослин. Блок схема алгоритму роботи системи зображена на рис. 3.1.

Рисунок 3.1 – Блок-схема роботи алгоритму системи

Принцип роботи системи полягає в аналізі зображень, отриманих з камер, які розташовані у теплиці. Після підключення мікроконтролера до відеопотоку нейромережа починає обробляти отримані дані. На першому етапі проводиться ідентифікація об'єктів на зображенні для визначення, чи присутні на ньому

рослини. У випадку, якщо рослини виявлено, система переходить до наступного етапу, де здійснюється розпізнавання їх видів. На основі цього розпізнавання нейромережа аналізує стан кожної рослини, перевіряючи її на можливі відхилення від нормального стану.

Якщо система фіксує відхилення, проводиться додатковий аналіз, щоб з’ясувати їхню причину. Визначається, чи є ці відхилення природними та зумовленими сезонними змінами, чи вони викликані захворюваннями рослин. У разі підтвердження хвороби система автоматично генерує сповіщення і відправляє його до Telegram-каналу, що дозволяє оперативно інформувати відповідальних осіб про проблему.

Додатковою перевагою цього процесу є можливість не тільки виявляти хвороби, а й аналізувати подібні випадки у базі даних, щоб надати рекомендації щодо їхнього усунення. Це підвищує ефективність управління теплицею, дозволяючи швидко реагувати на потенційні загрози та запобігати поширенню захворювань серед рослин. Система також враховує природні фактори, що дозволяє уникати хибних тривог і зосереджуватись на реальних загрозах для здоров’я рослин.

3.2 Вибір та обґрунтування апаратних рішень АПК

Вибираючи придатний одноплатний комп'ютер для моделювання АПК, треба було враховувати багато різних деталей. По-перше треба приділити увагу вартості так як найдешевші одноплатні комп'ютери коштують близько 10 доларів США, а дорогі і потужні плати для розробників до кількох сотень доларів.

Для простих завдань, які не вимагають великих обчислювальних ресурсів, підходить практично будь-яка плата. Однак для багатозадачності чи обробки великих обсягів даних необхідно використовувати більш потужний пристрій.

3.2.1 Raspberry Pi 4

Raspberry Pi 4 є популярним вибором серед одноплатних комп'ютерів завдяки оптимальному співвідношенню між цінлю та продуктивністю. Хоча існують інші потужніші плати, Raspberry Pi 4 залишається найкращим варіантом для більшості користувачів завдяки своїм доступним варіантам пам'яті у 2, 4 або 8 ГБайт і ціні, що починається з 35 доларів. Це багатофункціональний комп'ютер, розміром з банківську картку, що підходить для людей з різним рівнем кваліфікації завдяки своєї доступності та універсальності. Raspberry Pi Foundation пропонує багато офіційних ресурсів, таких як форуми та операційна система Raspberry Pi OS на базі Linux. Крім того, є велика підтримка від спільноти, зокрема сторонні книги, апаратні аксесуари, операційні системи та численні онлайн-форуми, блоги та спільноти (рис 3.2).

Рисунок 3.2 – Raspberry Pi 4 Model B [21]

Raspberry Pi 4 є дуже універсальним пристроєм, що дозволяє виконувати безліч різних завдань. Він чудово підходить для використання в якості заміни настільного комп'ютера для написання текстів, редагування зображень чи подкастингу. Також на його основі можна створити медіа-сервер, який буде цілком конкурентоспроможним, або зібрати дешевий файловий сервер, який може служити альтернативою дорогим мережевим пристроям зберігання даних (NAS). Raspberry Pi 4 також чудово підходить для створення розумних домашніх центрів, серверів для управління 3D-друком чи домашніх кінотеатрів (HTPC) за допомогою таких програм, як Kodi.

Однак, незважаючи на його широкі можливості, Raspberry Pi 4 не здатний справитися з усіма завданнями, такими як складна обробка великих обсягів даних або дуже ресурсоємні програми. Проте, він добре задокументований і може бути використаний для багатьох різноманітних проектів. Наприклад, можна запускати на ньому Linux, грати в ігри через RetroPie, створювати роботів на базі цього комп'ютера або розробляти проекти зі штучним інтелектом [21].

Плюси:

- універсальність;
- доступна ціна;
- варіативність вибору оперативної пам'яті;
- легкий для використання;
- чудове співвідношення ціни та продуктивності;
- маленький розмір;
- низьке енергоспоживання.

3.2.2 Одноплатний комп’ютер UDOO Bolt

UDOO Bolt один з найпотужніших одноплатним комп'ютером, завдяки своїм високим характеристикам і можливостям. Хоча він займає більше місця порівняно з Raspberry Pi та іншими компактними SBC, його продуктивність виправдовує цей розмір. UDOO Bolt побудований на архітектурі x86, що робить його особливим для тих хто шукає сумісність з операційними системами Windows 10 та різними дистрибутивами Linux, такими як Ubuntu та Debian. Це робить його відмінним вибором для ресурсомістких завдань, таких як обробка відео, 3D-моделювання та інші складні обчислення, де потрібна більша обчислювальна потужність, ніж може запропонувати Raspberry Pi (рис. 3.3).

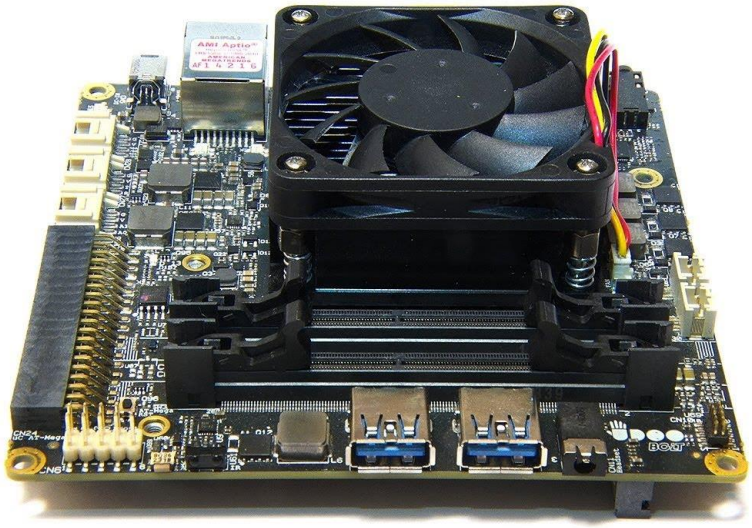


Рисунок 3.3 – UDOO BOLT V8 [22]

UDOO Bolt представлений у двох версіях – V3 і V8, обидві з яких побудовані

на базі передової технології AMD Ryzen, що забезпечує високу продуктивність і багатофункціональність. UDOO Bolt V3 оснащений двоядерним процесором AMD Ryzen V1202B із базовою тактовою частотою 2,3 ГГц, яку можна підвищити до 3,2 ГГц, а також інтегрованою графікою AMD Radeon Vega 3, що гарантує стабільну роботу з мультимедіа та іншими задачами. UDOO Bolt V8, в свою чергу, має чотирьоядерний процесор AMD Ryzen V1605B із базовою частотою 2 ГГц і можливістю розгону до 3,6 ГГц, доповнений потужною графікою Radeon Vega 8, яка забезпечує високу продуктивність для складних обчислень і графічно насичених завдань.

Розмір Bolt більший, ніж у Raspberry Pi 4, але це компенсується вражаючими характеристиками. З підтримкою до 32 Гбайт оперативної пам'яті DDR4 у форматі SO-DIMM, SATA, та 32 Гбайт вбудованої пам'яті eMMC, цей одноплатний комп'ютер фактично є заміною настільному ПК. UDOO Bolt V8 може відтворювати сучасні AAA-ігри та емулювати консолі, як-от PlayStation 3 і Wii U, що робить його ідеальним для ігрових і мультимедійних задач.

Ціни на Bolt починаються від понад 300 доларів для версії V3 і перевищують 400 доларів для V8. Хоча це недешевий варіант, його продуктивність, яка майже вдвічі перевищує показники Apple MacBook Pro 13", виправдовує вартість. Завдяки архітектурі x86 і сумісності з Windows та Linux, а також можливостям запуску високопродуктивних додатків, UDOO Bolt із SATA є потужним і надійним рішенням для користувачів, які шукають компактну альтернативу настільному комп’ютеру [22].

3.2.3 Міні-ПК Nvidia Jetson Xavier NX

Одноплатний комп'ютер Nvidia Jetson Xavier NX є високопродуктивним пристроєм, здатним виконувати широкий спектр завдань, особливо у сфері розробки зі штучним інтелектом. Основою його потужності є 384-ядерний графічний процесор Nvidia Volta із тензорними ядрами, які забезпечують ефективну обробку алгоритмів глибокого навчання. Крім того, Jetson Xavier NX оснащений шестиядерним процесором ARMv8.2, що додає йому універсальності

для виконання як графічних, так і загальних обчислювальних задач (Рис 3.4).

Рисунок 3.4 – Nvidia Jetson Xavier NX [23]

Пристрій має 8 Гбайт оперативної пам’яті LPDDR4 із 128-бітною шиною, що забезпечує високу пропускну здатність для інтенсивних обчислень. Вбудований модуль eMMC на 16 Гбайт дозволяє зберігати операційну систему та необхідні програми без додаткового зовнішнього накопичувача. Два інтегрованих двигуни Nvidia Deep Learning Accelerator (NVDLA) значно підвищують його здатність до виконання завдань, пов’язаних із машинним навчанням.

Nvidia Jetson Xavier NX також вражає своїми мультимедійними можливостями, включаючи підтримку відтворення відео у форматі 4K через два виходи DisplayPort/HDMI. Завдяки розширеним можливостям вводу/виводу, серед яких USB 3.1 та GPIO, цей одноплатний комп'ютер ідеально підходить для проектів, що вимагають підключення до різноманітного периферійного обладнання [23].

3.2.4 Висновок з вибору одноплатних комп’ютерів

Вибір одноплатних комп’ютерів для розподіленої системи health-моніторингу теплиць базувався на ключових критеріях, таких як вартість, продуктивність, розмір, архітектура, підтримувана операційна система та специфічні вимоги до обчислювальних ресурсів.

Raspberry Pi 4 було визначено як оптимальне рішення завдяки його збалансованості між ціною та функціональністю. Цей пристрій підтримує виконання широкого спектра завдань, включаючи обробку зображень і реалізацію базових алгоритмів машинного навчання. Його популярність обумовлена доступністю, низьким енергоспоживанням, різними конфігураціями оперативної пам'яті (2 ГБ, 4 ГБ, 8 ГБ) і широкою підтримкою спільноти.

Для задач, що потребують високої обчислювальної потужності, таких як обробка відео чи навчання складних нейронних мереж, кращими варіантами є UDOO Bolt та Nvidia Jetson Xavier NX. UDOO Bolt, завдяки архітектурі x86 та процесорам AMD Ryzen, забезпечує продуктивність, порівнянну з настільними ПК, і підходить для ресурсоємних застосунків. Nvidia Jetson Xavier NX із графічним процесором Volta і тензорними ядрами спеціалізується на завданнях глибокого навчання та штучного інтелекту, забезпечуючи високу ефективність для проектів у цій галузі.

Таким чином, вибір конкретного одноплатного комп’ютера залежить від складності поставлених завдань: для стандартних потреб обирається Raspberry Pi 4, а для більш інтенсивних обчислень – UDOO Bolt чи Nvidia Jetson Xavier NX.

3.3 Завантаження і налаштування програмного середовища і тренування нейромережі

Для успішної реалізації задачі розпізнавання зображень рослин та їхніх хвороб, необхідно підготувати відповідне програмне середовище та набір даних за допомогою якого буде проходити тренування. Мовою програмування для розподіленої системи обрано Python. Ця мова ідеально підходить для програмування на одноплатних комп’ютерах.

Python – це мова програмування, яка відома своєю простотою та гнучкістю. Вона була створена для того, щоб полегшити розробку програм і роботу з кодом. Вона підходить як для новачків, так і для досвідчених програмістів завдяки зрозумілому синтаксису та широкому набору бібліотек.

Python можна використовувати для різних завдань, від створення веб-сайтів до наукових досліджень, аналізу даних та штучного інтелекту. Її популярність також зростає завдяки великій підтримці з боку спільноти та розвитку безлічі корисних інструментів, які значно спрощують роботу розробників [24].

Для тренування моделі нейронної мережі було вирішено взяти “PlantVillage” який є великим і відкритий набіром даних, який був створений в рамках ініціативи PlantVillage для досліджень у сфері машинного навчання та комп'ютерного зору з метою покращення здоров'я рослин. Набір даних містить тисячі зображень рослин, що мають різні захворювання, а також зображення здорових рослин. Це важливий ресурс для розробки алгоритмів і моделей штучного інтелекту, які допомагають у діагностиці хвороб рослин [25].

Для того, щоб почати працювати з TensorFlow і використовувати датасет PlantVillage для тренування моделей машинного навчання, треба спочатку встановити TensorFlow наступною командою

pip install tensorflow

Наступним кроком є встановлення бібліотеки Kaggle за допомогою якої ми завантажимо потрібний нам датасет:

!kaggle datasets download -d abdallahalidev/plantvillage-dataset

Після розпакування датасету можна перевірити його структуру та вивести основну інформацію, щоб зрозуміти, як організовані файли (рис. 3.5).

```
print(os.listdir("plantvillage dataset"))


print(len(os.listdir("plantvillage dataset/segmented")))
print(os.listdir("plantvillage dataset/segmented")[:5])

print(len(os.listdir("plantvillage dataset/color")))
print(os.listdir("plantvillage dataset/color")[:5])

print(len(os.listdir("plantvillage dataset/grayscale")))
print(os.listdir("plantvillage dataset/grayscale")[:5])
```

```
['grayscale', 'segmented', 'color']
38
['Tomato___Spider_mites Two-spotted_spider_mite', 'Tomato___Bacterial_spot', 'Soybean___healthy', '
38
['Tomato___Spider_mites Two-spotted_spider_mite', 'Tomato___Bacterial_spot', 'Soybean___healthy', '
38
['Tomato___Spider_mites Two-spotted_spider_mite', 'Tomato___Bacterial_spot', 'Soybean___healthy', '
```

Рисунок 3.5 – Перевірка класів датасету

Це виведе список всіх підкаталогів у датасеті. Датасет **PlantVillage** містить три основні каталоги: **color** – зображення в кольоровому форматі, **grayscale** – зображення в градаціях сірого, **segmented** – сегментовані зображення (де рослини виділені на фоні).

Для подальшої роботи треба завантажити одне зображення з датасету та перевірити форму та значення пікселів (рис. 3.6).

Рисунок 3.6 – Отримання даних про зображення

Далі задаємо генератори, які будуть відповідати за подачу даних в модель:

```
# Train Generator
train_generator = data_gen.flow_from_directory(
base_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    subset='training',
    class_mode='categorical')
```

Наступним кроком є побудова моделі яка складається з кількох згорткових (convolutional) та пулингових (pooling) шарів, а також повнозв'язних (fully connected) шарів (рис. 3.7).

```
# Model Definition
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(img_size, img_size, 3)))
model.add(layers.MaxPooling2D(2, 2))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D(2, 2))


model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(train_generator.num_classes, activation='softmax'))
```

Рисунок 3.7 – побудова моделі нейронної мережі

Виводимо підсумок моделі, який показує розміри кожного шару та кількість параметрів, що тренуються. Це означає, що модель має понад 47 мільйонів параметрів, які будуть тренуватися (рис. 3.8).

```
# model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2  (None, 111, 111, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 54, 54, 64)        0
 g2D)

 flatten (Flatten)           (None, 186624)            0

 dense (Dense)               (None, 256)               47776000

 dense_1 (Dense)             (None, 38)                9766

=================================================================
Total params: 47805158 (182.36 MB)
Trainable params: 47805158 (182.36 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Рисунок 3.8 – Перевірка тренувальних параметрів

Після цього будується графік точності моделі, що порівнює точність на навчальному наборі та на валідаційному наборі протягом епох. Графік показує зміну точності на навчальному та валідаційному наборах, що допомагає виявити, чи є переобучення (рис. 3.9).

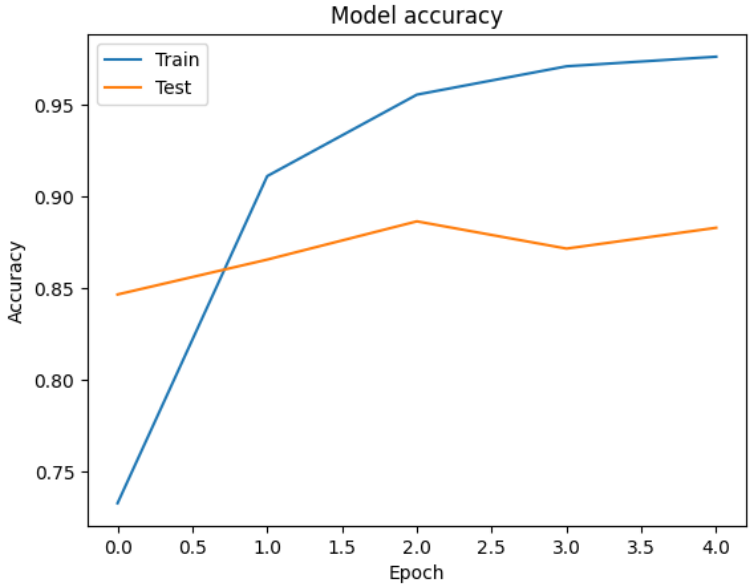


Рисунок 3.9 – графік точності моделі

Також треба побудувати графік втрат моделі який графік ілюструє зміну втрат на навчальному та валідаційному наборах і дозволяє оцінити, чи є покращення в результатах моделі з часом.

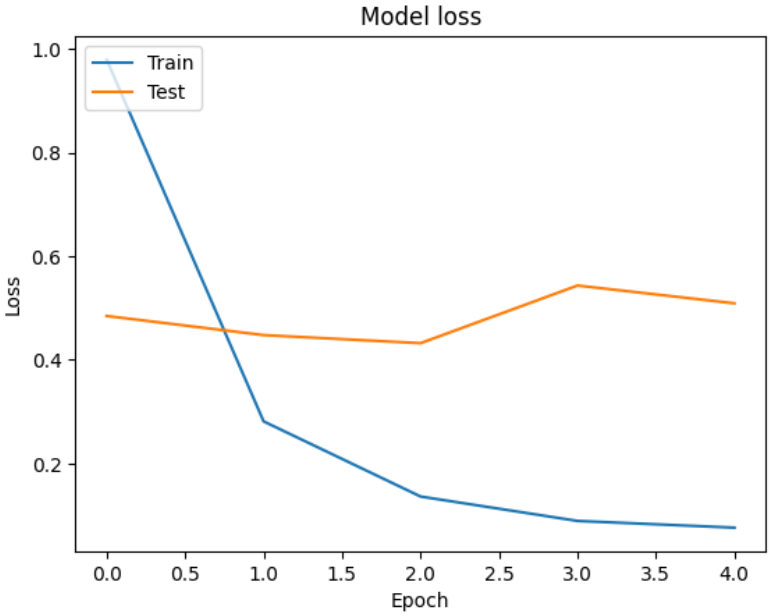


Рисунок 3.10 – Графік втрат моделі

Графіки потрібні для того, щоб наочно відслідковувати процес навчання моделі. Вони дозволяють швидко побачити, як точність і втрата змінюються під час тренування, що допомагає виявити проблеми, як переобучення чи недонавчання.

Такі візуалізації дають зрозуміти, чи модель адекватно узагальнює на нові дані, та допомагають коригувати стратегію навчання для досягнення кращих результатів.

Далі йде функція завантаження та попередньої обробки зображення яка завантажує зображення з вказаного шляху, змінює його розмір до заданих розмірів (за замовчуванням 224 x 224 пікселів), конвертує зображення в масив NumPy.

```
def load_and_preprocess_image(image_path, target_size=(224, 224)):
    img = Image.open(image_path)  # Завантаження зображення
    img = img.resize(target_size)  # Зміна розміру
    img_array = np.array(img)  # Перетворення в масив
    img_array = np.expand_dims(img_array, axis=0)  # Додавання виміру пакета
    img_array = img_array.astype('float32') / 255.  # Масштабування
    return img_array
```

І останнє, що треба зробити, це отримання прогнозу від моделі для зображення, яке вона отримає, і вибір найбільш ймовірного результату, що означає пошук класу з найвищою ймовірністю в масиві прогнозів. Цей результат дозволяє точно визначити, до якого класу належить зображення, на підставі прогнозу моделі.

```
def predict_image_class(model, image_path, class_indices):
    preprocessed_img = load_and_preprocess_image(image_path)
    predictions = model.predict(preprocessed_img)
    predicted_class_index = np.argmax(predictions, axis=1)[0]
    predicted_class_name = class_indices[predicted_class_index]
    return predicted_class_name
```

На цьому можна вважати нейромережу натренованою і переходити до перевірки її роботи. Після підтвердження задовільних результатів модель можна зберегти для подальшого використання на нових даних або в інтеграціях з реальними додатками.

Висновки до розділу 3

У третьому розділі було розглянуто алгоритм функціонування розподіленої системи health-моніторингу теплиць, що дозволяє ефективно виявляти захворювання рослин у режимі реального часу. Особливу увагу приділено вибору апаратних компонентів, таких як одноплатні комп’ютери, які є ключовими для роботи системи. Зокрема, проаналізовано, що Raspberry Pi 4 завдяки своїй доступній ціні та достатньому рівню продуктивності широко використовується для загальних завдань моніторингу. Водночас UDOO Bolt і Nvidia Jetson Xavier NX виявилися оптимальними для сценаріїв, які потребують високої обчислювальної потужності, наприклад, у роботі з машинним навчанням або обробкою великих обсягів даних.

Розробка програмного забезпечення здійснювалася з використанням мови програмування Python, що забезпечує гнучкість і широкий вибір бібліотек для роботи з нейронними мережами. Фреймворк TensorFlow використовувався для створення та тренування моделей глибокого навчання. Для навчання нейронної мережі було застосовано набір даних PlantVillage, який містить тисячі зображень рослин із різними типами захворювань, що дозволило створити надійну модель для розпізнавання хвороб. Описано процес тренування моделі, який включає побудову архітектури, аналіз точності та втрат, а також підготовку до застосування у реальних умовах.

4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ. АНАЛІЗ РЕЗУЛЬТАТІВ РОЗРОБКИ

4.1 Користувацький гід для системи оповіщення

Для того щоб зрозуміти коли буде знайдено хворобу у рослини за допомогою нейромережі треба налаштувати систему сповіщення на мобільний пристрій. У цьому нам допоможе телеграм бот за допомогою якого ми будемо відправляти повідомлення про хвороби одразу коли вони будуть виявлені.

У Telegram реалізовано надзвичайно простий і зручний механізм надсилання повідомлень за допомогою API ботів. Жодна інша платформа не забезпечує такої легкості та гнучкості в організації передачі повідомлень до чату або каналу "з боку". Всі повідомлення, надіслані через бота, автоматично синхронізуються та доставляються всім підписникам бота незалежно від пристрою: вони одночасно відображаються як на смартфонах, так і на комп’ютерах. Завдяки цьому Telegram забезпечує надійність і швидкість доставки сповіщень.

Більше того, Telegram дозволяє реалізувати двосторонню взаємодію. За допомогою API ви можете не лише відправляти сповіщення від пристрою до користувача, але й приймати команди від користувача та передавати їх пристрою. Це відкриває можливості для створення інтерактивних IoT-рішень, де людина може керувати обладнанням через Telegram, перетворюючи звичайний чат на інтерфейс управління.

Першим кроком є створення бота і налаштування каналу куди він буде надсилати повідомлення. Завдяки тому що в telegram є вбудовані функції для створення нових ботів то це робиться за декілька команд (рис 4.1).

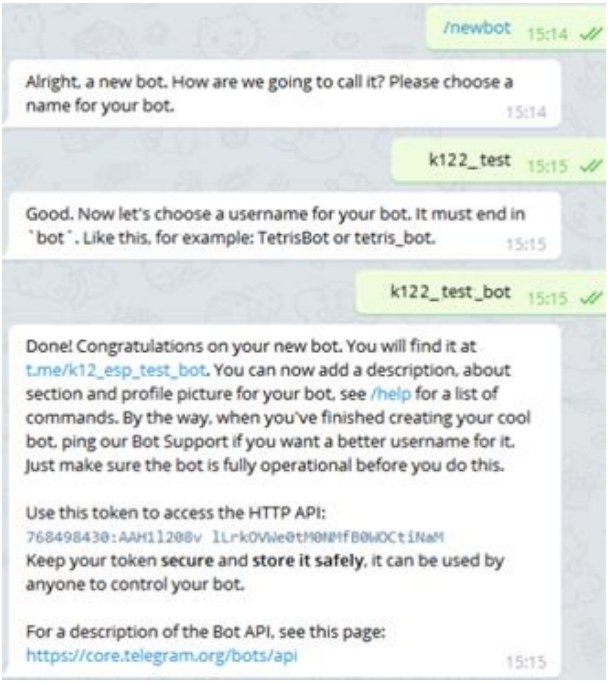


Рисунок 4.1 – Створення нового бота

Після створення ми копіюємо токен доступу до API, який знадобиться для подальшої роботи з ним, і створюємо скрипт для відправлення повідомлень з одноплатного комп'ютера через бота (рис 4.2).

```
#if defined(rSerialDebug)
Serial.printf("TLG :: Send message \"%s\": ", message.c_str());
#endif
// Формируем повідомлення
device.replace(" ", "%20");
message.replace(" ", "%20");
timestamp.replace(" ", "%20");
    String request = String("<b>") + device + "</b>" + "%0D%0A%0D%0A" + message +
        "%0D%0A%0D%0A<code>" + timestamp + "</code>";
  // Відправляємо HTTP-запит
if (tgClient.connect(tg_ApiServer, httpsPort)) {
  tgClient.printf(tg_ApiPost, request.c_str());
  tgClient.printf(tg_ApiHost, tg_ApiServer);
  tgClient.println(F("User-Agent: ESP8266 (nothans)/1.0"));
  tgClient.println(F("Connection: close"));
  tgClient.println();
```

Рисунок 4.2 – Фрагмент коду відправлення повідомлення

Крім власне функції надсилання повідомлень, у бібліотеку вбудована черга повідомлень, що надсилаються. Це потрібно це для того, щоб можна було помістити в чергу повідомлення коли немає підключення до WiFi, а вже після того, як підключення буде автоматично відновлено повідомлення з черги буде успішно надіслано.

```
bool tgSend(String device, String message) {
```

```
            if (_queueTg.push(new pubTgMessage(device, message))) {
            #if defined(rSerialDebug)
            Serial.printf("TLG :: Message \"%s\" :: added to queue, total %d
messages\n", message.c_str(), _queueTg.count());
            #endif    }
          else {
            #if defined(rSerialDebug)
            Serial.printf("TLG :: failed post message '%s' - queue is full!\n",
message.c_str());
            #endif    };};
```

Залишилося завантажити даний скрипт на плату і протестувати роботу повідомлення відправивши декілька тестових повідомлень (рис 4.3).

Рисунок 4.3 – Тестове повідомлення надіслане за допомогою мікроконтролера

Вийшла дуже зручна і гнучка система оповіщення.Крім того, параметри оповіщення можна легко налаштовувати засобами самого клієнта Telegram, а також налаштувати умови отримання повідомлень для певного чату чи групи. Це дає змогу створити персоналізований і зручний канал комунікації між Raspberry Pi і користувачами.

4.2 Налаштування одноплатного комп'ютера

Першим і ключовим елементом, необхідним для роботи одноплатного комп'ютера Raspberry Pi, є операційна система (ОС). Для цього пристрою доступний широкий вибір ОС, включаючи Raspbian (який тепер має назву Raspberry Pi OS), Ubuntu, LibreELEC, Windows 10 IoT Core та інші. Більшість із них базуються на ядрі Linux, що забезпечує сумісність із численними інструментами, необхідними для роботи з сучасними технологіями, зокрема засобами штучного інтелекту (ШІ).

Установка Raspberry Pi OS значно спрощена завдяки використанню офіційного інструменту «Raspberry Pi Imager». Ця утиліта, доступна на офіційному сайті Raspberry Pi, забезпечує користувачу інтуїтивно зрозумілий інтерфейс для вибору операційної системи та носія для запису. Зокрема, за допомогою Raspberry Pi Imager можна швидко обрати потрібну ОС, а також вказати диск (SD-карту або USB-накопичувач), на який буде записана система (рис. 4.4).

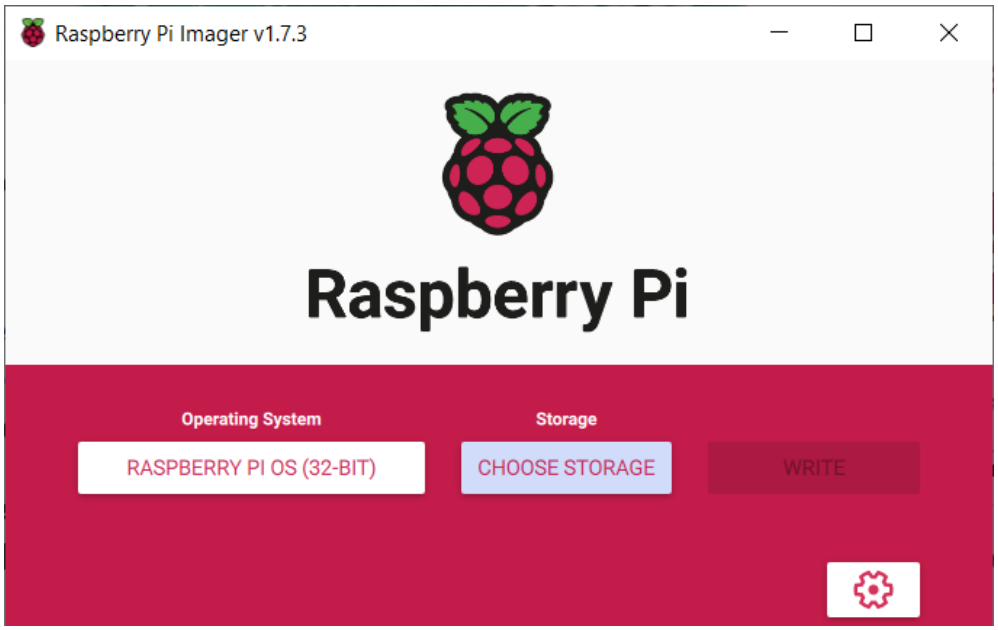


Рисунок 4.4 – Інтерфейс Raspberry Pi Imager

Коректна робота застосунків, що використовують штучний інтелект (ШІ), зазвичай вимагає постійного підключення до мережі Інтернет для доступу до хмарних ресурсів, моделей або оновлень. На щастя, Raspberry Pi оснащений вбудованим бездротовим інтерфейсом Wi-Fi, що забезпечує зручне підключення до мережі.

Налаштування Wi-Fi-з’єднання можна здійснити за допомогою вбудованого програмного забезпечення Raspberry Pi OS – raspi-config. Цей інструмент надає простий інтерфейс для конфігурації системи, включаючи мережеві параметри. (Рис 4.5).

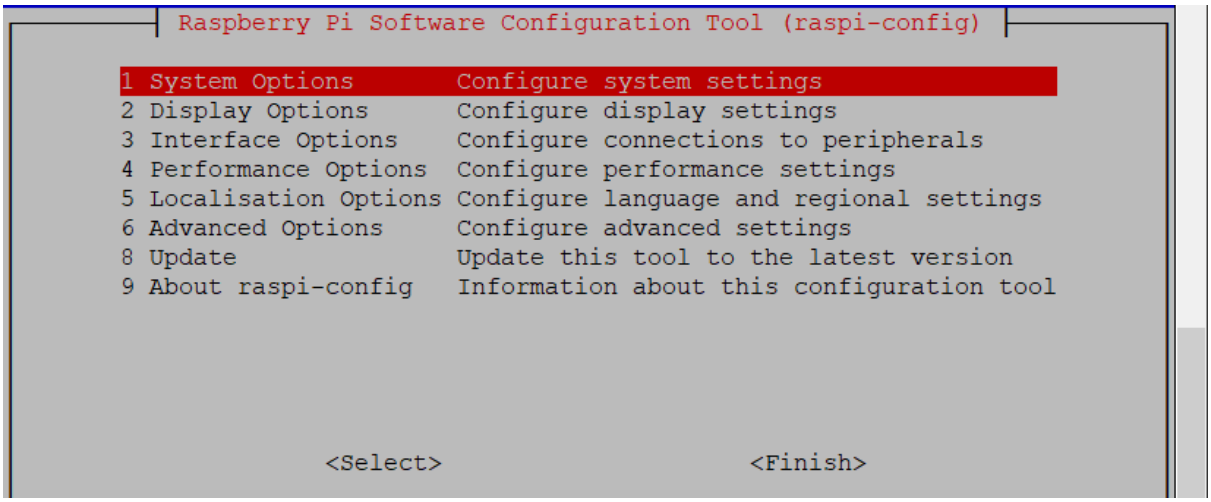


Рисунок 4.5 – Інтерфейс Raspi-config

Перевірка коректності налаштувань мережі є важливим кроком для забезпечення стабільного підключення до Інтернету. У Raspberry Pi OS це можна здійснити за допомогою команди «ip a». Ця команда виводить перелік усіх доступних мережевих інтерфейсів і їхні поточні налаштування. Вбудований бездротовий адаптер Raspberry Pi у списку адаптерів позначається як «wlan0». Якщо адаптер «wlan0» отримав IP-адресу це свідчить про те, що налаштування були виконані коректно і пристрій підключений до мережі (рис. 4.6).

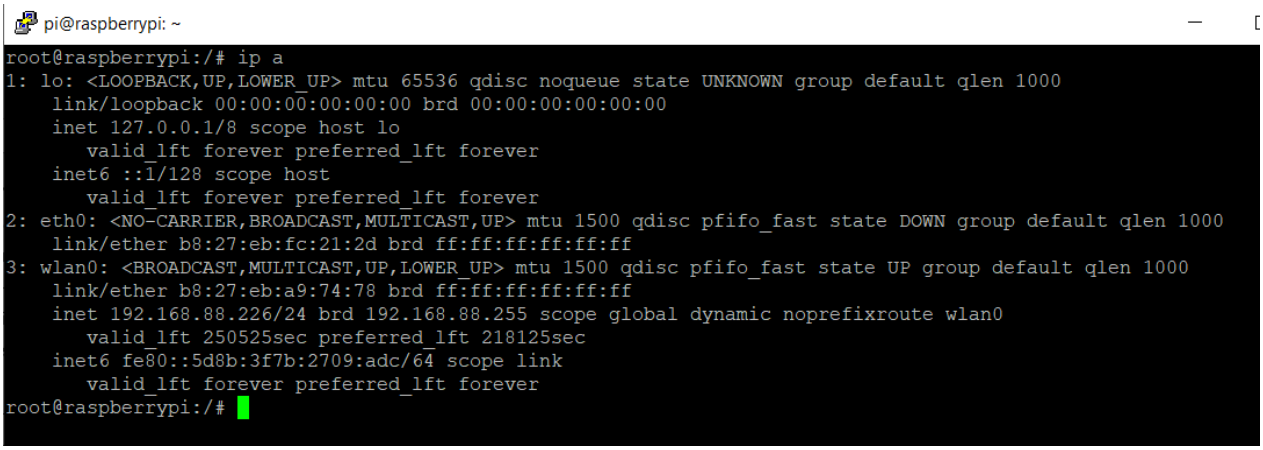


Рисунок 4.6 – Налаштування підключення до мережі

Налаштування одноплатного комп'ютера Raspberry Pi виходить далеко за межі конфігурації бездротової мережі. Для реалізації програмного забезпечення з використанням інструментів штучного інтелекту (ШІ) часто потрібне

завантаження та встановлення додаткових бібліотек і залежностей. Це можна легко здійснити за допомогою менеджера керування пакунками.

В операційній системі Raspberry Pi OS для цього використовується Advanced Packaging Tool (APT) – потужний інструмент для управління програмним забезпеченням. Він дозволяє завантажувати, встановлювати, оновлювати та видаляти пакунки з офіційних репозиторіїв, забезпечуючи доступ до широкого спектра бібліотек і утиліт для ШІ..

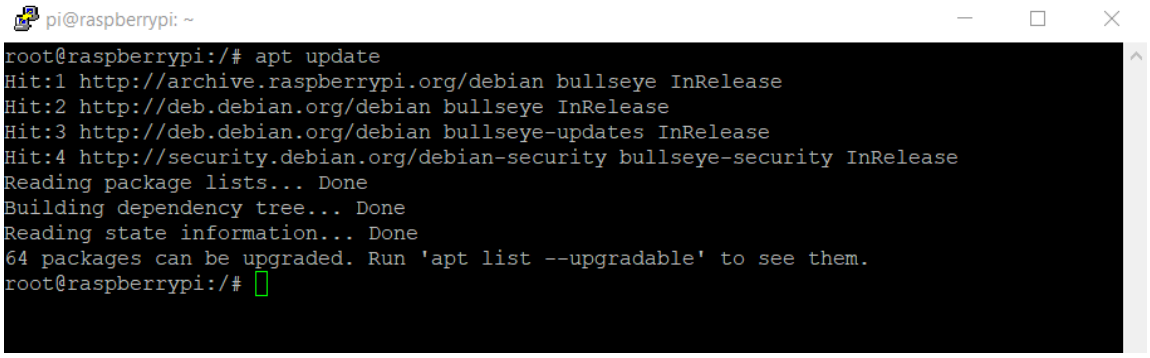


Рисунок 4.7 – Використання інструменту керування пакунками APT

Мова програмування Python забезпечує високий рівень зручності для розробників, зокрема завдяки можливості створення віртуальних оточень. Цей механізм дозволяє забезпечити зворотну сумісність між різними версіями Python і бібліотек, що є особливо важливим для роботи з проєктами, які мають специфічні вимоги до середовища виконання.

Додати до оточення нову бібліотеку можливо власним менеджером керування пакунками Python – pip3. Встановлення засобів ШІ відбувається схожим чином.

pip3 install tensorflow

Далі треба перемістити натреновану нейромережу на Raspberry Pi яку попередньо зберегти у форматі .Keras за допомогою команди:

model.save('plant_disease_prediction_model.keras')

Передати модель на Raspberry Pi можна декількома способами в нашому випадку за допомогою SCP бо Raspberry Pi і комп'ютер знаходяться в одній мережі і ми знаємо його IP-адресу.

scp plant_disease_prediction_model.keras pi@192.168.1.50:/home/pi/

Наступним кроком є створення Python-скрипта для передбачення зображення та виведення інформації про хвороби з подальшим оповіщенням, який фактично залишився незмінним з тренування моделі, лише з додаванням функції для відправлення повідомлення через телеграм (рис 4.8).

```python
# Виведення результату в консоль
print(f"Predicted class: {predicted_class}")

# Функція для відправлення повідомлення через Telegram
def send_message(message):
    try:
        bot = telegram.Bot(token=TOKEN)
        bot.send_message(chat_id=CHAT_ID, text=message)
        print("Повідомлення надіслано успішно!")
    except Exception as e:
        print(f"Помилка: {e}")

# Формування повідомлення для Telegram
message = f"Передбачена хвороба рослини: {predicted_class}"

# Відправлення повідомлення
send_message(message)
```

Рисунок 4.8 – Функція відправлення повідомлення

Розроблена модель здатна прогнозувати хворобу рослини на основі зображень, отриманих за допомогою камер. Після аналізу зображення, модель генерує передбачення щодо типу хвороби та автоматично надсилає результат у Telegram, де користувач отримує детальну інформацію про виявлену хворобу. Цей процес дозволяє швидко і точно отримувати діагнози без необхідності вручну перевіряти стан рослин, що робить систему надзвичайно корисною у галузі сільського господарства.

4.3 Тестування ефективності роботи готової системи

Тестування готового програмного забезпечення буде здійснюватися у два етапи. Перший етап передбачає перевірку роботи нейронної мережі на

правильність визначення хвороби у рослин. На цьому етапі буде оцінюватися точність моделі за різних умов, зокрема з використанням зображень та відео різної якості. Для більшої зручності та детальнішої перевірки буде використовуватися простий веб-інтерфейс, що дозволить завантажувати зображення хворих рослин і аналізувати результати роботи нейромережі.

Другий етап тестування спрямований на перевірку коректності надходження повідомлень. Буде проаналізовано, наскільки швидко та точно система інформує користувача про результати обробки даних. Це включатиме перевірку механізмів сповіщення, їх надійності та відповідності вимогам проєкту.

Обидва етапи тестування спрямовані на забезпечення повної відповідності роботи програмного забезпечення заявленим вимогам, а також на виявлення можливих недоліків, які можуть вплинути на ефективність системи.\

4.3.1 Тестування правильності визначення хвороби

Для початку будемо використовувати веб-інтерфейс для отримання результату для зручності та швидкості роботи. Завантаживши декілька фото рослин різної якості та пару відео використаємо навчену модель для виявлення хвороб (рис 4.9).

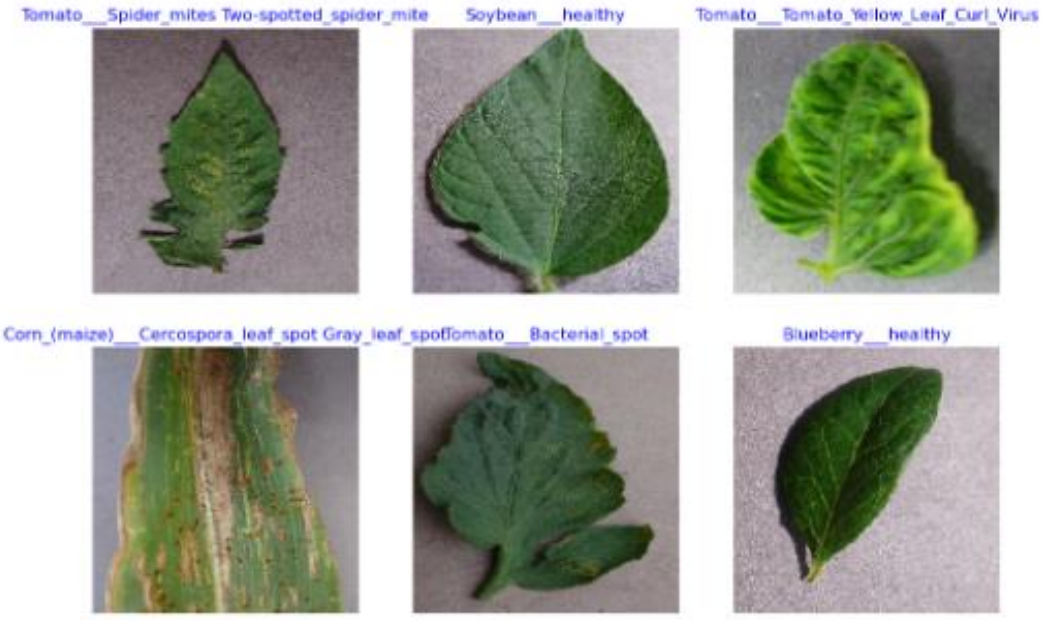


Рисунок 4.9 – Результати тестування за допомогою фото

Рисунок 4.10 – Результати тестування за допомогою відео

Рисунок 4.11 – Результати тестування за допомогою відео

Протестувавши дану систему можна сказати що вона правильно показує результати хвороби, але може помилятися про степінь ураження хворобою рослини через обмежену якість фотографій.

4.3.2 Результати роботи системи сповіщення

Для цього вже можна не використовувати веб-інтерфейс, а одразу надсилати зображення на перевірку нейромережі і вона коли виявить хворобу то надішле повідомлення про хворобу, для перевірки було використано фотографії і з хворобами і без них.

Рисунок 4.12 – Результат роботи системи сповіщення

Також треба протестувати функцію відправлення повідомлень при перебоях з інтернетом, тобто при втраті інтернет сигналу повідомлення будуть знаходитися в черзі та відправлять одразу при відновленні зв’язку (рис 4.13).

Рисунок 4.13 – Результат роботи функції черги повідомлень

З результатів досліджень, можна зробити висновок, що розроблена система на базі raspberry pi має високу точність у прогнозуванні результатів та має конкурентну спроможність з сучасними аналогами. До недоліків можна віднести те що для роботи треба підключення до інтернету та гарна якість зображення.

Висновки до розділу 4

У четвертому розділі присвячений опису експериментальні дослідження процесу налаштування, тестування та аналізу ефективності роботи системи health-моніторингу теплиць. Розроблено гід для інтеграції телеграм-бота для надсилання сповіщень, налаштуванні апаратного та програмного забезпечення, а також тестуванні точності діагностування хвороб рослин та роботи системи оповіщення.

Розроблений телеграм-бот забезпечує швидку і надійну передачу сповіщень про виявлені захворювання рослин. Він дозволяє отримувати повідомлення на різних пристроях, підтримує двосторонню взаємодію для передачі команд від користувачів до системи та функцію черги для надсилання повідомлень при нестабільному з’єднанні. Це створює зручний канал комунікації між користувачем і системою.

Налаштування Raspberry Pi включає встановлення операційної системи Raspberry Pi OS, конфігурацію бездротового підключення, інсталяцію додаткових бібліотек та перенесення натренованої моделі нейронної мережі. За допомогою Python та бібліотеки TensorFlow створено програмне середовище для аналізу стану рослин і надсилання сповіщень.

Тестування проводилося у два етапи: перевірка точності визначення хвороб за зображеннями та відео різної якості, а також аналіз швидкості і надійності надсилання повідомлень. Результати показали високу точність моделі в діагностуванні, хоча якість зображень може впливати на визначення ступеня ураження. Система оповіщення виявилася ефективною, забезпечуючи своєчасну доставку сповіщень навіть за умов тимчасової відсутності інтернет-з’єднання.

Цей підхід до інтеграції апаратного і програмного забезпечення, а також використання нейронної мережі, дозволяє автоматизувати моніторинг стану рослин у теплицях, підвищуючи ефективність сільськогосподарських процесів і знижуючи ризики втрат від захворювань.

Аналіз результатів показав, що розроблений АПК є ефективним, точним і зручним у використанні. Його функціональні можливості та якість роботи повністю відповідають визначеним вимогам. Використання бездротових технологій забезпечує значні переваги в умовах реального використання, підвищуючи мобільність і гнучкість комплексу. Окрім того, система має значний потенціал для подальшого вдосконалення та широкого впровадження у практичну діяльність, що відкриває перспективи її адаптації для різноманітних сфер застосування.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено апаратно-програмний комплекс (АПК) для моніторингу стану здоров'я рослин у теплицях, який базується на використанні сучасних технологій IoT та штучного інтелекту. Результати проведених досліджень і тестувань підтверджують, що створена система відповідає поставленим вимогам, забезпечує високу точність аналізу, зручність у використанні та оперативність у реагуванні на потенційні загрози.

На основі аналізу аналогів і існуючих рішень було обґрунтовано вибір апаратної платформи Raspberry Pi 4, яка завдяки інтегрованим модулям Wi-Fi, багатофункціональним GPIO-інтерфейсам та підтримці обчислювальних бібліотек є оптимальним рішенням для реалізації даного проєкту. Сформована специфікація вимог дозволила визначити цілі, задачі та компоненти, необхідні для створення системи моніторингу.

Проведено проєктування апаратної частини комплексу, створення структурної схеми та тестування з'єднань у програмі Fritzing. Розроблено програмну частину комплексу. Використання мови програмування Python та фреймворка TensorFlow дозволило ефективно реалізувати алгоритми обробки зображень, розпізнавання хвороб рослин та інтеграції з телеграм-ботом для надсилання сповіщень. Реалізовані механізми автоматичного збору даних, їх аналізу за допомогою нейронних мереж та інтерактивного оповіщення розширили функціональність комплексу, забезпечивши його готовність до використання.

У рамках роботи проведено тренування нейронної мережі на наборі даних PlantVillage, що включає тисячі зображень здорових і уражених рослин. Для забезпечення точності класифікації було використано методи аугментації даних, які значно підвищили стійкість моделі до змін умов освітлення та якості зображень. Модель досягла гарних показників точності у тестових середовищах, що підтверджує її ефективність тренування.

Тестування системи підтвердило її працездатність і високу точність діагностування захворювань рослин. У порівнянні з традиційними методами

моніторингу розроблений комплекс показав вищу ефективність і оперативність, що є ключовими перевагами для сільського господарства. Незначні затримки у передачі даних пов'язані з характером бездротових технологій і не впливають на загальну ефективність системи.

Розроблений АПК дозволяє аналізувати дані про стан рослин, що дає змогу виявляти закономірності в поширенні захворювань та адаптувати агротехнологічні процеси до нових умов. Це сприяє не лише запобіганню втратам, а й оптимізації витрат на обробку рослин, зменшенню використання хімічних засобів та підвищенню екологічності сільського господарства.

Таким чином, розроблений АПК для моніторингу здоров'я рослин у теплицях є сучасним, доступним та ефективним інструментом, який сприяє підвищенню продуктивності, запобіганню втратам через хвороби та підтримці екологічної стійкості. Його функціональність, точність і можливість подальшого вдосконалення відкривають перспективи для впровадження у реальну практику агротехнологій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Комп'ютерний зір. URL: https://www.facerua.com/pakiet-dlia-novachkiv-shcho-takie-kompiutiernii-zir/#:~ (Дата звернення: 05.10.2024).

2. A Gentle Introduction to Computer Vision. URL: https://machinelearningmastery.com/what-is-computer-vision/ (Last accessed: 07.10.2024).

3. Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN. URL: https://www.mathworks.com/help/vision/ug/getting-started-with-r-cnn-fast-r-cnn-and-faster-r-cnn.html (Last accessed: 07.10.2024).

4. ABBYY FineReader. URL: https://pdf.abbyy.com/finereader-pdf/#:~:text=FineReader%20PDF%20integrates%20scanned%20documents,documents%20in%20the%20digital%20workplace (Last accessed: 02.11.2024).

5. YOLO: Real-Time Object Detection. URL: https://pjreddie.com/darknet/yolov2/ (Last accessed: 07.10.2024).

6. Фото нейрона Історія обчислювальних нейронних мереж. URL: https://nagornyy.me/it/istoriia-vychislitelnykh-neironnykh-setei/ (Last accessed: 07.10.2024).

7. Neural Network fundamentals. URL: https://medium.com/datadriveninvestor/neural-network-fundamentals-1956a3000c24, вільний (Last accessed: 06.10.2024).

8. Гудфелов І. А. Глибоке навчання. *MIT Press*. 2016. – Бібліограф: с. 140 DOI: https://doi.org/10.15407/jai2021.01.032

9. A Deep Architecture: Multi-Layer Perceptron. URL: https://medium.com/@nlunge786/a-deep-architecture-multi-layer-perceptron-164bc5ff3842 (Last accessed: 07.10.2024).

10. Deep Learning for NLP (without Magic) - Richard Socher and Christopher Manning. URL: https://www.slideshare.net/slideshow/deep-learning-for-nlp-without-magic-richard-socher-and-christopher-manning/24777161 (Last accessed: 07.10.2024).

11. Deep Learning Neural Networks Explained in Plain English. URL: https://www.freecodecamp.org/news/deep-learning-neural-networks-explained-in-plain-english/ (Last accessed: 02.11.2024).

12. Neural network for facial feature extraction. URL: https://www.researchgate.net/figure/Neural-network-for-facial-feature-extraction_fig4_351219828 (Last accessed: 03.11.2024).

13. Хести Т. Елементи статистичного навчання. *Спрингер-Верлаг*, 2019. – 746 с. DOI: https://doi.org/10.30525/978-9934-26-084-1-11.

14. Harry Fairhead, Mike James. Raspberry Pi IoT In Python Using GPIO Zero. *Springer Nature*. 2020. P 220-229. ISBN 1871962668

15. Stanley Chiang. Hacking the System Design Interview: Real Big Tech Interview Questions and In-depth Solutions. 2022. P.130-150. ISBN 979-8839126497

16. Краснопоясовський А. С. Інформаційний синтез інтелектуальних систем керування: підхід, що ґрунтується на методі функціонально–статистичних випробувань. *СумДУ*. 2004. –261 с. DOI: doi: 10.33407/itlt.v8i4.101.

17. What is OpenCV Library. URL: https://www.geeksforgeeks.org/opencv-overview/ (Last accessed: 02.11.2024).

18. Mohamed Elgendy. Deep Learning for Vision Systems. *Simplified Chinese* 2020. P.142-144. ISBN 9781617296192

19. Thushan Ganegedara. TensorFlow in Action. 2022. P.245-249. ISBN 9781617298349

20. TensorFlow і машинне навчання. URL: https://foxminded.ua/tensorflow-shcho-tse/ (Дата звернення 03.12.2024).

21. Raspberry Pi 4 Model B 4GB. URL: https://arduino.ua/ru/prod3308-raspberry-pi-4-4gb (Дата звернення 03.12.2024).

22. UDOO BOLT. URL: https://www.udoo.org/discover-the-udoo-bolt/ (Last accessed: 02.12.2024).

23. NVIDIA Jetson Xavier NX Developer Kit URL: https://evo.net.ua/nvidia-jetson-xavier-nx-developer-kit (Last accessed: 02.12.2024).

24. What is Python. URL: https://www.teradata.com/glossary/what-is-python#:~:text=Python%20is%20used%20for%20server,dynamic%20typing%2C%20and%20dynamic%20binding. (Last accessed: 02.12.2024).

25. PlantVillage Dataset. URL: https://www.kaggle.com/datasets/emmarex/plantdisease (Last accessed: 02.12.2024).

ДОДАТОК А
КОД ПРОГРАМИ

Файл CNN

{
"cell_type": "code",
"source": [
"# saving the class names as json file\n",
"json.dump(class_indices, open('class_indices.json', 'w'))"
],
"metadata": {
"id": "StM3_I3UwjFV"
},
"execution_count": 29,
"outputs": []
},
{
"cell_type": "code",
"source": [
"# Example Usage\n",
"image_path = '/content/test_apple_black_rot.JPG'\n",
"#image_path = '/content/test_blueberry_healthy.jpg'\n",
"#image_path = '/content/test_potato_early_blight.jpg'\n",
"predicted_class_name = predict_image_class(model, image_path, class_indices)\n",
"\n",
"# Output the result\n",
"print(\"Predicted Class Name:\", predicted_class_name)"
],
"metadata": {

```
   "id": "kJb9gQGRw2Ln",
   "colab": {
    "base_uri": "https://localhost:8080/"
   },
   "outputId": "f329cc1c-2945-416a-f42d-174a433ff60c"
  },
  "execution_count": 31,
  "outputs": [
   {
    "output_type": "stream",
    "name": "stdout",
    "text": [
     "1/1 [==============================] - 0s 266ms/step\n",
     "Predicted Class Name: Apple___Black_rot\n"
    ]
   }
  ]
 },
 {
  "cell_type": "markdown",
  "source": [
   "**Save the model to Google drive or local**"
  ],
  "metadata": {
   "id": "QBkknsKMyDbs"
  }
 },
 {
  "cell_type": "code",
```

"source": [

"model.save('drive/MyDrive/Youtube/trained_models/plant_disease_prediction_model.h5')"

],
"metadata": {
"id": "OfoTNemcxjk5"
},
"execution_count": null,
"outputs": []
},
{
"cell_type": "code",
"source": [
"model.save('plant_disease_prediction_model.h5')"
],
"metadata": {
"id": "J8ByAMH6ykbN",
"colab": {
"base_uri": "https://localhost:8080/"
},
"outputId": "8836c7a9-6d35-421f-b36c-f6fb50fd5cf7"
},
"execution_count": 32,
"outputs": [
{
"output_type": "stream",
"name": "stderr",
"text": [

"/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.\n",
" saving_api.save_model(\n"
]
}
]
},
{
"cell_type": "code",
"source": [],
"metadata": {
"id": "ln01Rmj0L8Hg"
},
"execution_count": null,
"outputs": []
}
]
}

Файл QueueArray

```
#ifndef _QUEUEARRAY_H
#define _QUEUEARRAY_H
template<typename T>
class QueueArray {
  public:
    // init the queue (constructor).
    QueueArray (const uint16_t initialSize);
```

```
  // clear the queue (destructor).
  ~QueueArray ();

  // push an item to the queue.
  bool push (const T i);

  // pop an item from the queue.
  T pop ();

  // get an item from the queue.
  T peek () const;

  // check if the queue is empty.
  bool isEmpty () const;

  // get the number of items in the queue.
  uint16_t count () const;

  // check if the queue is full.
  bool isFull () const;

  // set the printer of the queue.
  void setStream (Stream & s);

private:

  // exit report method in case of error.
  void exit(const __FlashStringHelper*) const;
```

```
  // led blinking method in case of error.
  void blink () const;

  Stream * stream; // the printer of the queue.
  T * contents;    // the array of the queue.

  uint16_t size;       // the size of the queue.
  uint16_t items;      // the number of items of the queue.

  uint16_t head;       // the head of the queue.
  uint16_t tail;       // the tail of the queue.
};

// init the queue (constructor).
template<typename T>
QueueArray<T>::QueueArray (const uint16_t initialSize) {
  size = 0;       // set the size of queue to zero.
  items = 0;      // set the number of items of queue to zero.

  head = 0;       // set the head of the queue to zero.
  tail = 0;       // set the tail of the queue to zero.

  stream = NULL; // set the printer of queue to point nowhere.

  // allocate enough memory for the array.
  contents = (T *) malloc (sizeof (T) * initialSize);

  // if there is a memory allocation error.
  if (contents == NULL)
```

```
    exit (F("QUEUE: insufficient memory to initialize queue."));

  // set the initial size of the queue.
  size = initialSize;
}

// clear the queue (destructor).
template<typename T>
QueueArray<T>::~QueueArray () {
  free (contents); // deallocate the array of the queue.

  contents = NULL; // set queue's array pointer to nowhere.
  stream = NULL;  // set the printer of queue to point nowhere.

  size = 0;        // set the size of queue to zero.
  items = 0;       // set the number of items of queue to zero.

  head = 0;        // set the head of the queue to zero.
  tail = 0;        // set the tail of the queue to zero.
}

// push an item to the queue.
template<typename T>
bool QueueArray<T>::push (const T i) {
  // check if the queue is full.
  if (isFull ())
    // we cannot add anythif - just return false
    return false;
```

```
  // store the item to the array.
  contents[tail++] = i;

  // wrap-around index.
  if (tail == size) tail = 0;

  // increase the items.
  items++;

  //ok everything was fin
  return true;
}

// pop an item from the queue.
template<typename T>
T QueueArray<T>::pop () {
  // check if the queue is empty.
  if (isEmpty ())
    exit (F("QUEUE: can't pop item from queue: queue is empty."));

  // fetch the item from the array.
  T item = contents[head++];

  // decrease the items.
  items--;

  // wrap-around index.
  if (head == size) head = 0;
```

```
  // return the item from the array.
  return item;
}

// get an item from the queue.
template<typename T>
T QueueArray<T>::peek () const {
  // check if the queue is empty.
  if (isEmpty ())
    exit (F("QUEUE: can't peek item from queue: queue is empty."));

  // get the item from the array.
  return contents[head];
}

// check if the queue is empty.
template<typename T>
bool QueueArray<T>::isEmpty () const {
  return items == 0;
}

// check if the queue is full.
template<typename T>
bool QueueArray<T>::isFull () const {
  return items == size;
}

// get the number of items in the queue.
template<typename T>
```

```
uint16_t QueueArray<T>::count () const {
  return items;
}
// set the printer of the queue.
template<typename T>
void QueueArray<T>::setStream (Stream & s) {
  stream = &s;
}
// exit report method in case of error.
template<typename T>
void QueueArray<T>::exit (const __FlashStringHelper * m) const {
  // print the message if there is a printer.
  if (stream)
    stream->println (m);
  // lop blinking until hardware reset.
  blink ();
}
// led blinking method in case of error.
template<typename T>
void QueueArray<T>::blink () const {
  while(1);
  // solution selected due to lack of exit() and assert().
}
#endif // _QUEUEARRAY_H
```

Файл tg_bot

```
class pubTgMessage {
  public:
    pubTgMessage(String _device, String _message) {
      device = _device;
```

```
        message = _message;
      };
    String device;
    String message;
  };

  QueueArray<struct pubTgMessage*> _queueTg(queueSizeTg);
  bool tgSend(String device, String message) {
      if (_queueTg.push(new pubTgMessage(device, message))) {
      #if defined(rSerialDebug)
      Serial.printf("TLG :: Message \"%s\" :: added to queue, total %d messages\n", message.c_str(), _queueTg.count());
      #endif
     }
     else {
      #if defined(rSerialDebug)
      Serial.printf("TLG :: failed post message '%s' - queue is full!\n", message.c_str());
      #endif
     };
  };
  bool tgSendEx(String device, String message) {
    bool result = false;
    if (WiFi.status() == WL_CONNECTED) {
        #if defined()
      WiFiClientSecure wifiClient;
      wifiClient.setCACert(rootCACertificate);
      AxTLS::WiFiClientSecure wifiClient;
     #endif
```

```
#if defined(rSerialDebug)
Serial.printf("TLG :: Send message \"%s\": ", message.c_str());
#endif

device.replace(" ", "%20");
message.replace(" ", "%20");
  String httpRequest = tg_ApiURL + String("<b>") + device + "</b>" + "%0D%0A%0D%0A" + message;
HTTPClient https;
if (https.begin(wifiClient, httpRequest)) {
 int httpCode = https.GET();
 if (httpCode > 0) {
  if (httpCode == HTTP_CODE_OK || httpCode == HTTP_CODE_MOVED_PERMANENTLY) {
   result = true;
        #if defined(rSerialDebug)
   Serial.println(httpCode);
   #endif
  }
  else {
   #if defined(rSerialDebug)
   Serial.printf("[ERROR] %d, %s\n", httpCode, https.errorToString(httpCode).c_str());
   #endif
  };
 }
 else {
  #if defined(rSerialDebug)
  Serial.println("[ERROR] Unable to connect");
```

```
      #endif
     };

     https.end();
    }
    else {
     #if defined(rSerialDebug)
     Serial.println("[ERROR] Unable to create HTTPS client");
     #endif
    };
  };
  return result;
}
bool tgProcessQueue()
{
  bool result = false;
    while ((WiFi.status() == WL_CONNECTED) && (!_queueTg.isEmpty()))
  {
    pubTgMessage *nextPub = _queueTg.peek();
    if (tgSendEx(nextPub->device, nextPub->message)) {
     result = true;
     nextPub = _queueTg.pop();
     delete nextPub;
    }
    else return result;
       delay(100);
  };
  return result;
}
```

ДОДАТОК Б

АПРОБАЦІЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили
ДНУ «Інститут модернізації змісту освіти»
Південний науковий центр НАН та МОН
Інститут української археографії та джерелознавства
імені М. С. Грушевського НАН України
Первинна профспілкова організація ЧНУ ім. Петра Могили

**«МОГИЛЯНСЬКІ ЧИТАННЯ – 2024:
досвід та тенденції розвитку суспільства в Україні:
глобальний, національний та регіональний аспекти»**

XXVII Всеукраїнська науково-практична конференція

ТЕЗИ ДОПОВІДЕЙ

ТЕХНІЧНІ НАУКИ

Миколаїв, 6–10 листопада 2024 року

Миколаїв – 2024

Список використаних джерел

1. Using the ICS43432 MEMS microphone on a Raspberry Pi with i2s. URL: https://github.com/nejohnson2/rpi-i2s (дата звернення: 05.10.2024).
2. Liu L., Liu M., Guo Q., Liu D., Peng Y. MEMS sensor data anomaly detection for the UAV flight control subsystem. *2018 IEEE SENSORS Proc.* New Delhi, India, 2018. P. 1–4. DOI: 10.1109/ICSENS.2018.8589748.
3. Квєтний Р. Н. , Богач І. В., Бойко О. Р., Софина О. Ю., Шушура О. М. Комп'ютерне моделювання систем та процесів. Методи обчислень. Ч. 1 : навч. посіб. Вінниця : ВНТУ, 2012. 193 с.
4. Фесенко О. Д., Бєляков Р. О., Радзівілов Г. Д. Імітаційне моделювання безплатформної інерціальної навігаційної системи БПЛА на основі нейромережевих алгоритмів. *Системи і технології зв'язку, інформатизації та кібербезпеки.* 2022. № 2 (2). С. 63–69. DOI: 10.58254/viti.2.2022.09.63.

УДК 004.89:631.548

Кисельов Д. М.,
магістрант,
Пузирьов С. В.,
канд. фіз.-мат. наук, доцент, доцент кафедри комп'ютерної інженерії,
ЧНУ імені Петра Могили, м. Миколаїв, Україна

РОЗПОДІЛЕНА СИСТЕМА HEALTH-МОНІТОРИНГУ ТЕПЛИЦЬ

Сучасне сільське господарство стикається з багатьма викликами, зокрема з необхідністю ефективного моніторингу здоров'я рослин у теплицях для забезпечення сталого розвитку виробництва та зниження втрат врожаю. Використання розподілених систем моніторингу з інтеграцією нейронних мереж та камер відеоспостереження пропонує революційні рішення для виявлення хвороб рослин на ранніх стадіях. Ці системи автоматизують процеси виявлення проблем, що дозволяє оптимізувати ресурси та підвищити продуктивність.

Ця робота присвячена розробці та впровадженні розподіленої системи health-моніторингу теплиць. Головною метою є створення ефективної системи моніторингу стану рослин у теплицях, що базується на використанні камер відеоспостереження та нейронних мереж, для

Підсекція:
➢ **КОМП'ЮТЕРНА ІНЖЕНЕРІЯ**