НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) Інформаційних технологій

УДК

ПОГОДЖЕНО
Декан факультету (Директор ННІ)
Інформаційних технологій
(назва факультету (ННІ))
_____ Глазунова О.Г., д.пед.н, проф.
(підпис) (ПІБ)
“___” _____20_р.

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
Комп'ютерних систем, мереж та кібербезпеки
(назва кафедри)
_____ Лахно В.А., д.т.н., проф
(підпис) (ПІБ)
“___” _____20_р.

МАГІСТЕРСЬКА РОБОТА

на тему Дослідження продуктивності комп’ютерної системи розпізнавання об’єктів у веб-середовищі

Спеціальність 123 «Комп’ютерна інженерія»
(код і назва)

Освітня програма Магістр
(назва)

Орієнтація освітньої програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Керівник магістерської роботи
к.т.н., доцент _____ Шкарупило В.В.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Ткаленко Д.С.
(підпис) (ПІБ студента)

КИЇВ – 2021

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) Інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних систем, мереж та кібербезпеки
д.т.н., проф. Лахно. В.А.
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"___" _____20____року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ РОБОТИ СТУДЕНТУ

Ткаленко Даніл Сергійович
(прізвище, ім'я, по батькові)

Спеціальність комп'ютерна інженерія
(код і назва)

Освітня програма магістр
(назва)

Орієнтація освітньої програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Тема магістерської роботи Дослідження продуктивності комп'ютерної системи розпізнавання об'єктів у веб-середовищі

затверджена наказом ректора НУБіП України від 23.10.2020р. № 1578 "С"

Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до магістерської роботи: комп'ютерна система розпізнавання об'єктів у веб-середовищі

Перелік питань, що підлягають дослідженню:

1. продуктивність комп'ютерної системи розпізнавання об'єктів
2. параметрі тренування моделі
3. сучасні засоби створення веб-додатків

Перелік графічного матеріалу (за потреби) _____

Дата видачі завдання "____" _____20___р.

Керівник магістерської роботи _____ Шкарупило В.В.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Ткаленко Д.С.
(підпис) (прізвище та ініціали студента)

РЕФЕРАТ

ПЗ: 69 с., 27 рис., 8 формул, 4 додатки, 15 джерел.

HTML, CSS, JS, МАШИННЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, РОСЛИНИ

Об’єкт розробки – комп’ютерна система розпізнавання об’єктів у веб-середовищі.

Мета роботи – дослідження системи розпізнавання зображень з використанням машинного навчання та сучасних веб-технологій.

Проект складається з чотирьох розділів.

Перший розділ присвячено аналізу проблеми діагностики хвороб рослин та огляду технологій машинного навчання як інструмента для їх вирішення.

У другому розділі наведено етапи проектування системи, а саме вибір технологій для реалізації проекту, планування архітектури додатку, проектування інтерфейсу та блок-схеми роботи системи.

У третьому розділі описано процес підключення та навчання нейронної мережі, а також описано етапи реалізації веб-системи, додано лістинги з кодом частин системи.

У четвертому розділі досліджено вплив різних параметрів на результати навчання моделі, проведено тестування отриманої моделі а також веб-інтерфейсу, наведені результати тестування.

У результаті виконання дипломної роботи було спроектовано, розроблено та досліджено систему класифікації зображень рослин з використання нейронної мережі та веб-інтерфейсу.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

Batches - групи прикладів, які використовуються для навчання мережі

CNN - згорткова нейронна мережа, Convolutional Neural Network

CSS - таблиці стилів, Cascading Style Sheets

Dropout - метод регуляризації для запобігання перенавчання мережі

DL - глибоке навчання, Deep Learning

HTML - мова розмітки, HyperText Markup Language

JS - мова програмування ДжаваСкрипт, JavaScript,

NN - нейронна мережа, Neural Network

Pooling - операція об'єднання в згорткових нейронних мережах

PWA - прогресивний додаток, Progressive Web Application

SLP - одношаровий перцептрон, Single-Layer Perceptron

SPA - односторінковий додаток, Singe Page Application

SSR - серверний рендерінг, Server-Side Rendering

TL навчання з учителем, навчання, Supervised Learning transfer learning

RGB - червоний, зелений, синій, Red, Green, Blue

ЗНМ - згорткова нейронна мережа

ВСТУП

На виробництво сільськогосподарських культур впливають різні фактори, такі як кліматичні умови, стан ґрунту, різні захворювання тощо. Існуючий метод виявлення хвороб рослин – це просто спостереження неозброєним оком, що вимагає більше людської праці, належно обладнаних лабораторій, дорогих приладів тощо. А неправильне виявлення хвороб може призвести до недосвідченого використання пестицидів, що може спричинити розвиток довгостроковоі стійкості патогенів, знижуючи здатність культури боротися. Виявити захворювання рослин можна, спостерігаючи за плямою на листках ураженої рослини.

Сучасні технології дали людському суспільству можливість виробляти достатньо їжі, щоб задовольнити потреби понад 7 мільярдів людей. Однак продовольча безпека залишається під загрозою через низку факторів, включаючи зміну клімату, зменшення кількості запилювачів, хвороби рослин, та інші. Хвороби рослин становлять не тільки загрозу продовольчій безпеці в глобальному масштабі, але також можуть мати катастрофічні наслідки для дрібних фермерів, чиї засоби до існування залежать від здорових культур. У країнах, що розвиваються, понад 80 відсотків сільськогосподарського виробництва виробляється дрібними фермерами, і повідомлення про втрати врожаю понад 50% через шкідників і хвороб є поширеними. Більше того, найбільша частка голодуючих (50%) живе в дрібних фермерських господарствах, що робить дрібних фермерів групою, яка особливо вразлива до збоїв у постачанні їжі, викликаних патогенами.

Для запобігання втраті врожаю через хвороби були розроблені різні заходи. Історичні підходи до широкого застосування пестицидів в останнє десятиліття все частіше доповнювалися підходами інтегрованої боротьби зі шкідниками. Незалежно від підходу, правильне визначення захворювання, коли воно вперше з'являється, є важливим кроком для ефективного лікування

захворювання. Історично виявлення хвороб підтримувалося сільськогосподарськими організаціями з поширення інформації або іншими установами, такими як місцеві клініки рослин. Останнім часом такі зусилля були додатково підтримані наданням інформації для діагностики захворювань в Інтернеті, що сприяє зростанню проникнення Інтернету в усьому світі. Ще зовсім недавно інструменти, засновані на мобільних телефонах, поширилися, використовуючи переваги історично безпрецедентного швидкого поширення технологій мобільних телефонів у всіх частинах світу.

Смартфони, зокрема, пропонують дуже нові підходи для виявлення захворювань через їхню обчислювальну потужність, дисплеї з високою роздільною здатністю та великий набір вбудованих аксесуарів, таких як передові HD-камери. За широкими оцінками, до 2022 року у світі буде від 5 до 6 мільярдів смартфонів. Поєднання факторів широкого поширення смартфонів, камер високої чіткості та високопродуктивних процесорів у мобільних пристроях призводять до ситуації, коли діагностика захворювання на основі автоматичного розпізнавання зображень, якщо це технічно можливо, може бути доступною в безпрецедентному масштабі. Ця робота націлена дослідження використання нейронних мереж для вирішення проблеми діагностики хвороб рослин з огляду на наведені вище фактори.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Огляд проблемної області та актуальність

Хвороби рослин стали основною загрозою для глобальної продовольчої безпеки. Хвороби рослин приносять 10–16% втрат у світовому врожаї сільськогосподарських культур щороку, що коштує приблизно 220 мільярдів доларів США. Згідно зі звітом Продовольчої та сільськогосподарської організації , у 2050 році населення нашої планети досягне 9,1 мільярда. Таким чином, сільськогосподарське виробництво необхідно збільшити до 70%, щоб задовольнити потреби в їжі в умовах постійно зростаючого населення. З іншого боку, рясне використання хімічних речовин, таких як бактерициди , фунгіциди та нематициди, для боротьби з хворобами рослин спричиняє несприятливий вплив на агроекосистему. В даний час існує потреба в ефективних методах раннього виявлення хвороб для боротьби з хворобами рослин для забезпечення продовольчої безпеки та стійкості агроекосистеми.

Хвороби рослин впливають на якість фруктів, овочів, зернових, бобових і спричиняють великі втрати у виробництві. Смертельні хвороби рослин призводять до високої смертності рослин. Наприклад, хвороба кокосової пальми каданг-каданг (Cocus nucifera L.) викликає передчасне занепад і загибель кокосових пальм. Повідомляється, що з моменту першого опису в 1914 році він знищив понад 40 мільйонів кокосових пальм у центральних Філіппінах. Як правило, хвороби рослин пошкоджують фотосинтетичний апарат і впливають на ріст рослин. Більшість хвороб рослин (близько 85%) викликані грибковими або грибоподібними організмами. Інші серйозні захворювання рослин викликаються бактеріями, віруси, і віроїди , і деякі захворювання викликаються певними нематодами.

Патогенні мікроорганізми поширені повсюдно. Патогени характеризують симптоми у рослин і викликають хвороби через сприйнятливість рослини до несприятливого впливу патогенів. Більшість

патогенів здійснюють важливу діяльність у природі, отримують харчування від хазяїна та зв'язуються з рослинами через симбіотичні чи несимбіотичні відносини. Перед дослідженням у лабораторії підозрілі рослини необхідно ідентифікувати за зовнішніми ознаками плодів і листям на плодах і листках. У більшості випадків ці видимі симптоми зазвичай проявляються на середніх і пізніх стадіях інфекції. Однак морфологічна ідентифікація захворювань не є надійною. Для виявлення збудника потрібен відповідний метод.

Нейронні мережі є найбільш перспективним інструментом для виконання цього завдання. Механізм НМ заснований на нервовій системі людини. В цілому, нейронні мережі дуже корисні для розпізнавання образів, незалежно від будь-яких явних правил розпізнавання, вимагають менш формальної статистики і здатні моделювати складні нелінійні зв'язки. Для вирішення проблеми виділення ознак в зображеннях з метою їх класифікації використовуються різноманітні методи формування набору ознак, які дозволяють чітко ідентифікувати зображення, тобто віднести їх до певної категорії. У більшості випадків для вирішення завдання виділення ознак на зображеннях листків рослин для класифікації типів хвороб використовуються особливості текстури зображень листків.

Далі ми розглянемо принцип роботи нейронних мереж для класифікації зображень більш детально.

Штучні нейронні мережі

1.2.1 Принципи побудови нейронних мереж

Штучна нейронна мережа (ШТМ) – сукупність моделей біологічних нейронних мереж. Це мережа елементів – штучних нейронів, з'єднаних синаптичними зв'язками. Мережа обробляє вхідну інформацію і формує набір вихідних сигналів у процесі зміни свого стану з часом.

Нейронні мережі виникли в результаті досліджень в області штучного інтелекту, тобто спроби відтворити здатність біологічної нервової системи навчатися і виправляти помилки шляхом моделювання низькорівневої структури мозку. Вперше математичну модель штучних нейронів запропонували В. Маккалох і В. Піттс. У 1958 році Френк Розенблат фактично реалізував мережу як комп'ютерну програму, яка пізніше була реалізована як електронний пристрій - персептрон.

Спочатку нейрони могли оперувати тільки сигналами логічних нулів і логічних одиниць, оскільки в їх основі лежить біологічний прототип, який може бути тільки в двох станах - збудженому або незбудженому. Розглянемо будову біологічних і штучних нейронів і взаємозв’язок між ними.

Мозок складається з дуже великої кількості (приблизно 10,000,000,000) нейронів, з'єднаних численними зв'язками. Нейрони – це спеціальні клітини, здатні поширювати електрохімічні сигнали. Тіло клітини містить безліч відростків, що гілкуються, двох типів-дендрити і аксони. Дендрити служать вхідними каналами для нервових імпульсів з інших нейронів. Ці імпульси надходять у тіло клітини розміром від 3 до 100 мікрон, викликаючи її специфічне збудження, яке поширюється аксоном. Аксони клітини з'єднуються з дендритами інших клітин за допомогою синапсів. При активації нейрон надсилає електрохімічний сигнал за своїм аксоном. Через синапс цей сигнал досягає інших нейронів, які можуть активуватися. Нейрон активується тоді, коли сумарний рівень сигналів, які у його ядро з дендритів, перевищить певний рівень (поріг активації). На рисунку 1.1 показано структуру біологічного нейрона.

Рисунок 1.1 - Типова структура нейрона

Штучний нейрон імітує властивості біологічного нейрона. На вхід штучного нейрона надходить кілька сигналів, кожен із яких є виходом іншого нейрона. Кожен вхід збільшується на відповідну вагу, аналогічну синоптичній силі, і всі твори підсумовуються, визначаючи рівень активації нейрона. Модель штучного нейрона представлена на рисунку 1.2.

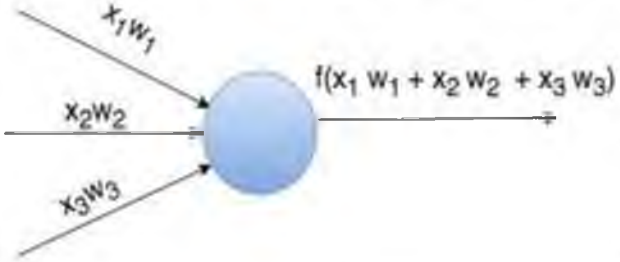


Рисунок. 1.2 - Модель штучного нейрона

Нейрон виконує кілька функцій:

- приймальна функція: синапси отримують інформацію;
- функція інтеграції: на виході нейронного сигналу, що несе інформацію про всі підсумовані нейронні сигнали;
- провідникова функція: інформація до синапсу проходить через аксон;
- передавальна функція: імпульс, який досяг кінця аксона, змушує

медіатор передавати збудження наступного нейрона.

Синапси - це зв'язки, за допомогою яких виходи одних нейронів досягають входів інших. Кожна комбінація має свою вагу. Зв’язки з позитивною вагою називаються збудливими, а негативні - гальмівними. Вихід нейрона називається аксоном. У штучній нейронній мережі штучний нейрон — це нелінійна функція, аргументом якої є лінійна комбінація всіх вхідних сигналів. Ця функція називається активацією. Тоді результат функції активації відноситься до виходу нейрона. З’єднуючи такі нейрони з іншими, отримують штучну нейронну мережу.

2 Активаційна функція

Одним із етапів розвитку нейронної мережі є вибір функції активації нейрона, яка використовується для генерування виходу нейрона. Активаційна функція нейрона характеризує зв'язок сигналу на виході нейрона з сумами сигналів на його входах. Частіше усього функція монотонно зростає і коливається між [-1,1] (гіперболічний тангенс) і [0,1] (сигмоїда). Для певних алгоритмів треиування важливо, щоб функція активації була диференційована по всій числовій осі. Для штучного нейрона характерна активуюча функція.

Існує кілька найпоширеніших функцій активації. Наприклад, функція порогової активації (функція Хевісайда). Його не можна використовувати для алгоритму зворотного поширення.

Сигмовидна функція активації є монотонно зростаючою, скрізь диференційованою, S-подібною нелінійною функцією з насиченням. Сигмовидна оболонка дозволяє підсилювати слабкі сигнали і не отримувати достатньо сильних сигналів. Сигмоїдальна активаційна функція виражається формулою:

За допомогою цієї функції можна отримати безперервне вихідне значення в діапазоні [0,1]. Крім того, він легко диференційований, що спрощує навчання мережі методом зворотного поширення помилки. Це найпоширеніша функція активації в багатьох нейронних мережах.

Гіперболічний тангенс (tanh) приймає будь-яке дійсне число як вхідні дані і виводить дійсне число від -1 до 1. Як і сигмовидна, гіперболічний тангенс може бути насиченим. Однак, на відміну від сигмовидної, вихід цієї функції має нульовий центр. Саме тому краще використовувати для тренування НМ гіперболічний тангенс, а не сигмоїд.

Гіперболічний тангенс:

В останні роки велику популярність придбала функція активації під назвою «випрямляч» (ReLU). Ця функція обчислюється за формулою:

і реалізує простий пороговий перехід в нулі.

Розрахунок сигмовидної та гіперболічної тангенс вимагає ресурсомістких операцій, таких як побудова до рівня, тоді як ReLU можна реалізувати шляхом простого перетворення порога матриці активації до нуля. Більше того, ReLU не схильний до насичення. Аналогічно, використання ReLU значно збільшує швидкість збіжності стохастичного градієнта порівняно з сигмоподібним та гіперболічним тангенсом. Вважається, що це пов’язано з лінійним характером і ненасиченням цієї функції. Цей тип функції активації найчастіше використовується в надточних шарах. Функція активації softmax є узагальненням логістичної функції для багатовимірного випадку. Функція перетворює вектор ності розмірності K у вектор $f(x)$ тієї ж розмірності, де

кожна координата $f(x)_i$ отриманого вектора представлена дійсним числом у проміжку [0,1] і сумою координат дорівнює 1.

Координати отриманого вектора інтерпретуються як ймовірність належності об'єкта до класу i.

Нейроині мережі для класифікації зображень

З появою великих обсягів даних та великих обчислювальних можливостей нейронні мережі почали активно використовувати. Особливу популярність отримали згорткові нейронні мережі, націлені на ефективне розпізнавання зображень. Свою назву архітектура мережі отримала через наявність операції згортки, суть якої в тому, що кожен фрагмент зображення множиться на матрицю (ядро) згортки поелементно - результат підсумовується та записується в аналогічну позицію вихідного зображення. В архітектуру мережі закладено фундаметальні знання з предметної області комп'ютерного зору: піксель зображення сильніше пов'язаний із сусіднім (локальна кореляція) та об'єкт на зображенні може зустрітися у будь-якій частині зображення.

Особливу увагу свічкові нейронні мережі отримали після конкусу ImageNet, який відбувся у жовтні 2012 року та був присвячений класифікації об'єктів на фотографіях. У конкурсі потрібно розпізнавання образів у 1000 категорій. Переможець цього конкурсу — Алекс Крижевський, використовуючи згорткову нейронну мережу, значно перевершив решту учасиників.

Успіх застосування згорткових нейронних мереж для класифікації зображень призвів до багатьох спроб використовувати цей метод до інших

завдань. Останнім часом їх почали активно використовувати для завдання класифікації текстів.

1.3.1 Архітектура згорткової нейронної мережі

Згорткова нейронна мережа зазвичай є чергуванням згорткових шарів, субдискретизуючих шарів і за наявності - повнозв'язних шарів на виході. Усі три види шарів можуть чергуватись у довільному порядку. У згортковому шарі нейрони, які використовують ті самі ваги, об'єднуються в карти ознак, а кожен нейрон карти ознак пов'язаний з частиною нейронів попереднього шару.

Рисунок. 1.3: Архітектура простої згорткової нейронної мережі

1.3.2 Субдискретизуючий шар

Шари цього типу виконують зменшення розмірності (зазвичай у кілька разів). Це можна робити різними способами, але найчастіше використовується метод вибору максимального елемента (max-pooling) — вся карта ознак поділяється на комірки , після чого вибираються максимальні за значенням.

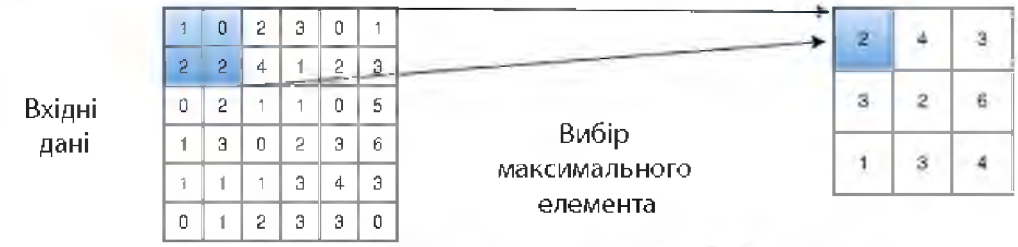


Рисунок 1.4 - Шар Субдискретізація

1.3.3 Згортковий шар

На відміну від повнозв'язкового, у згортковому шарі нейрон з'єднаний лише з обмеженою кількістю нейронів попереднього рівня, тобто згортковий

шар аналогічний застосуванню операції згортки, де використовується лише матриця ваг невеликого розміру (ядро згортки), яку «рухають» по всьому шару, що обробляється.

Основною характеристикою даного шару є так звані фільтри - багатовимірні матриці ваг зв'язку нейронів попереднього шару з нейронами згоркового шару. Ще одна особливість згоркового шару в тому, що він трохи зменшує зображення за рахунок крайових ефектів.

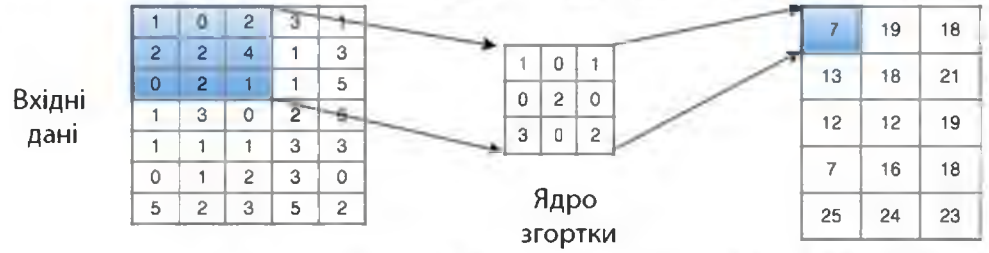


Рисунок 1.5 - Шар згортки

1.3.4 Повнозв'язний шар

Повнозв'язний шар є одновимірним. Якщо попередній шар має параметр глибини, кожен нейрон в ньому з'єднаний з кожним нейроном усіх рівнів у попередньому шарі. Основне призначення цього шару - перетворення сигналу, отриманого в згортковому шарі, в одновимірне представлення, а також виділення ознак в одновимірному шарі.

Цей шар також можна використовувати як останній (оригінальний) шар ЗНМ, а результатом є ймовірність приналежності до певного типу вхідного зображення (функція softmax).

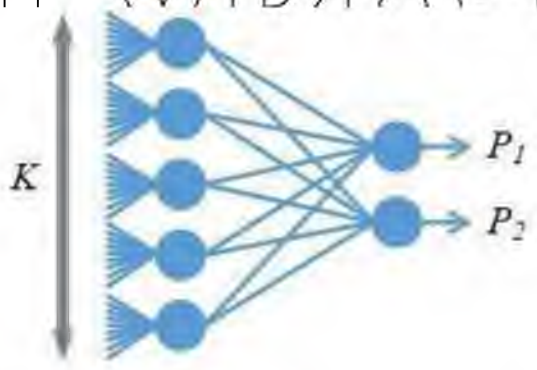


Рисунок 1.6 - Повнозв'язний шар

1.3.5 Dropout шар

Dropout шар - це спосіб боротьби з перенавчанням у нейронних мережах, навчання яких зазвичай виробляють стохастичним градієнтним спуском, випадково вибираючи деякі об'єкти з вибірки. Регуляризація Dropout полягає в зміні структури мережі: кожен нейрон викидається з деякою ймовірністю. По такій прорідженій мережі проводиться навчання, для ваг, що залишилися, робиться градієнтний крок, після чого всі викинуті нейрони повертаються в нейронну мережу. Таким чином, на кожному кроці стохастичного градієнта ми налаштовуємо одну з можливих 2N архітектур мережі, де під архітектурою ми розуміємо структуру зв'язків між нейронами, а через N позначаємо сумарне число нейронів. При тестуванні нейронної мережі нейрони не викидаються, але вихід кожного нейрона множиться на (1 − p) - завдяки цьому на виході нейрона ми будемо отримувати співставлення його відповіді по всіх 2N архітектур. Таким чином, навчену за допомогою dropout-регуляризації нейронну мережу можна розглядати як результат усереднення 2N мереж.

ПРОЕКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

Функції та призначення продукту

Фермери зазвичай виявляють хвороби посівів неозброєним оком, що змушує їх приймати важкі рішення щодо того, які добрива використовувати та як рятувати врожай. Це вимагає детальних знань про типи захворювань і великого досвіду, необхідного для того, щоб правильно визначати виявленні захворювання. Деякі хвороби, схожі між собою, можуть збити з пантелику навіть досвідченого фермера.

Щоб запобігти цій ситуації, ми можемо використовувати розглянуті вище технології розпізнавання та класифікації зображень за допомогою нейронних мереж, щоб правильно ідентифікувати хвороби рослин і видати більш стандартизовані результати.

Додаток, розроблений в ході роботи над даним дипломним проектом, є закликаним виконувати саме ці задачі. На основі спеціально зібраних наборів даних і за допомогою машинного навчання система може не тільки визначати види рослин, а й визначати стан здоров’я рослин. Якщо він покаже ознаки хвороби, система визначить, яка рослина дійсно хвора, і відобразить результати аналізу та поради фермерам. Всі функції програми включають в себе:

- реєстрація та авторизація у додатку;
- обрання типу рослини зображення якої будуть класифікуватися;
- підготовка та завантаження зображення у сервіс для класифікації;
- класифікація хвороби рослини з переліку підтримуваних хвороб;
- опис хвороб та рекомендації по догляду за рослинами.

2.2 Визначення архітектури проекту

Веб-програма - це програма клієнт-сервер, де є браузер (клієнт) і веб-сервер. Логіка веб-додатка розподілена між сервером і клієнтом, є канал для обміну інформацією та зберігання даних локально або в хмарі. Веб-додатки з'явилися як етап еволюції веб-сайту і, справді, мають багато спільного. Факторами, які відрізняють веб-сайт від веб-програми, є інтерактивність , інтеграція та аутентифікація . Перед початком розробки веб-додатків важливо вибрати тип архітектури веб-додатка та модель компонентів. Правильний вибір дуже важливий для успіху проекту.

Архітектура веб-додатків — це структура високого рівня, яка визначає спосіб функціонування, продуктивності та масштабування вашого продукту та бізнесу. Далі буде розглянуто актуальні варіанті рішень та обґрунтовано вибір певної архітектури для реалізації проекту.

2.2.1 SSR — server side rendering

Говорячи про основні принципи роботи в Інтернеті, ми зазвичай маємо на увазі архітектуру клієнт-сервер. Клієнт запитує вміст із сервера, де розташовані бізнес-логіка та база даних. Використовуючи простий JavaScript, статична веб-сторінка надсилає запит до служби (можливо, API). Служба повертає дані та відображає HTML-сторінку клієнту.

Якщо програма відображається на стороні сервера, вміст витягується з сервера і передається до браузера для відображення користувачеві. Якщо сторінка HTML відображається на стороні сервера, користувач повинен перейти на сторінку, перш ніж браузер отримає сторінку з сервера. Це означає, що для відображення вмісту користувачеві потрібно більше часу. Для кешування вмісту сторінки ця схема часто постачається з Nginx, веб-сервером, який також можна використовувати як проксі та балансувальник

навантаження.

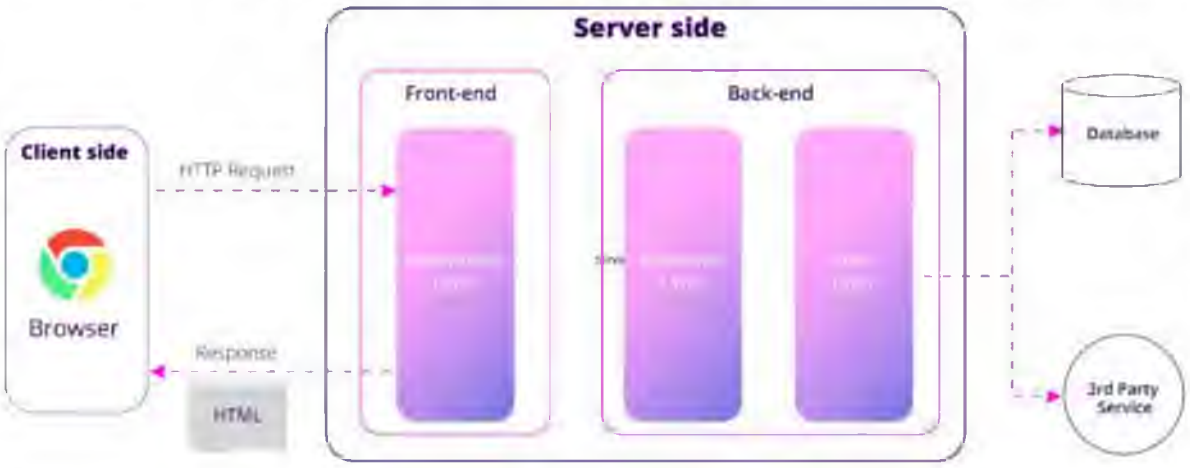


Рисунок 2.1 - Схема SSR архітектури

Той факт, що HTML відображається на сервері, забезпечує ряд переваг, таких як SEO, можливість посилань і миттєве завантаження першої сторінки. Відтворення на стороні сервера працює, коли JS вимкнено у браузері. При обробці коду на сервері ніяких особливих вимог до браузера не пред'являється – це дозволяє миттєво виявити помилки.

Однак SSR не може обробляти важкі запити сервера (повторні HTML, CSS), що призводить до повільного відтворення під час завантаження сервера або перезавантаження повної сторінки. Але справжньою ахіллесовою п'ятою цього базового типу архітектури веб-додатків є погана взаємодія з кінцевим користувачем і неможливість створити повноцінний інтерфейс користувача.

Іншими словами, SSR — це простий і економічно ефективний спосіб, якщо вам потрібно створити простий веб-сайт. Реалізація даного типу архітектури можлива за допомогою будь-якої мови програмування та серверної частини.

2.2.2 SGG — Генерація статичного сайту

Процес створення статичного сайту включає генератор, який автоматизує кодування окремих HTML-сторінок, створюючи їх із шаблонів. Вибираючи Static Site Generation, ви отримуєте простий статичний веб-сайт,

розташований на CDN або будь-якому сервері, який містить уже згенеровану сторінку HTML, яку можна надати користувачам за запитом. Тому не потрібно створювати його щоразу, коли хтось відвідує ваш веб-сайт — сервер просто надсилає вже наявні дані через API.

Перш за все, цей підхід підходить лише для веб-сайтів. Крім того, вміст створених сторінок веб-сайту не змінюється, якщо ви не додасте нові дані чи компоненти. Це означає, що вам доведеться повністю перегенерувати веб-сайт, як тільки ви захочете додати новий вміст. Це один із основних недоліків, який серйозно обмежує бізнес-кейсів, до яких він застосовний.

Серед переваг, однак, є висока швидкість статичного вмісту, який доставляється через CDN. Крім того, в SSG-всі операції з сервером і робота з базою даних реалізуються через API, незалежний від веб-сайту. Цей варіант простий і, таким чином, виключно доступний у реалізації.

SPA — Односторінковий додаток

SPA — це тип веб-додатка, який працює в браузері. Він не вимагає перезавантаження сторінки, коли потрібно відобразити нові дані. Цей тип архітектури веб-додатків широко використовується в нашому повсякденному житті: Facebook, Gmail, Google Maps, GitHub і Twitter – усі вони є односторінковими програмами.

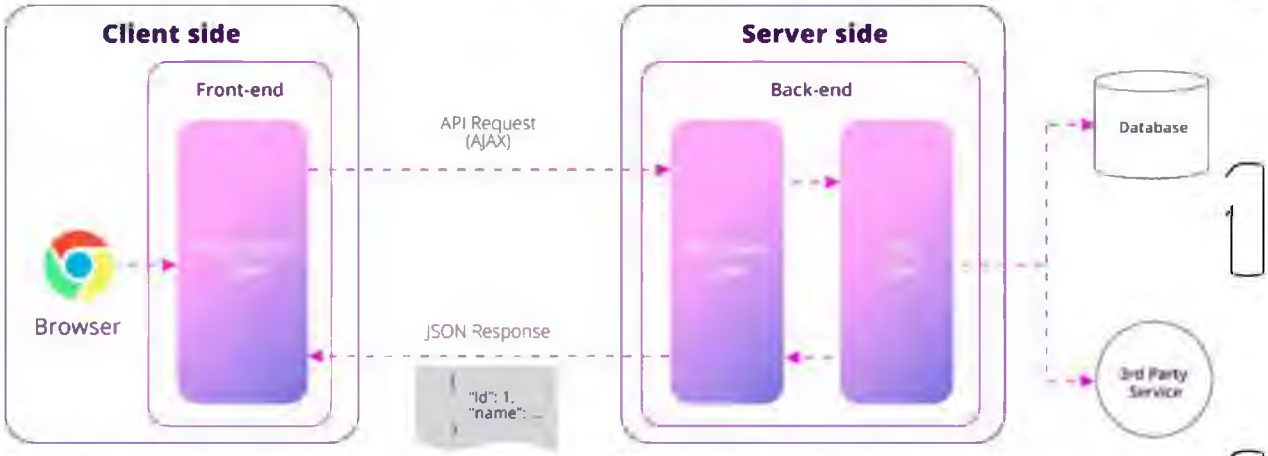


Рисунок 2.2 - Схема SPA

На відміну від SSR і SSG, SPA дозволяє створювати інтерактивний веб-додаток. Він використовує API для зв'язку з сервером. Ця архітектура добре підходить для легкого масштабування вашого продукту. Крім того, якщо потрібен мобільний додаток, для розробки API не потрібні додаткові зусилля – мобільний додаток може використовувати той самий API, що й веб.

SPA забезпечує швидке відтворення після повного завантаження програми у браузер і створює програмне забезпечення для кінцевого користувача з високою відповідністю. У той же час це «вбиває» ваше SEO та обмежує зв'язування, оскільки реалізація такого функціоналу потребує додаткових зусиль.

Серед інших недоліків – тривалий час, необхідний для першого завантаження, погана маршрутизація та обмежена підтримка застарілих браузерів. Будучи досить дорогим типом веб-архітектури, SPA підходить для створення адаптивного інтерфейсу користувача для B2C.

2.2.4 PWA — Прогресивний веб-додаток

Архітектура прогресивної веб-програми використовує логіку односторінкового веб-додатка з деякими службами, які запускаються далі, у браузері. Це означає, що головне, що слід враховувати, це те, що і браузер, і ОС повинні підтримувати цей набір стандартів.

Для кінцевого користувача прогресивний веб-додаток фізично означає спливаючу пропозицію додати програму на екран запуску (не браузер, а екран операційної системи), коли він відвідує веб-сайт. Якщо користувач погоджується, програма автоматично додається на пристрій.

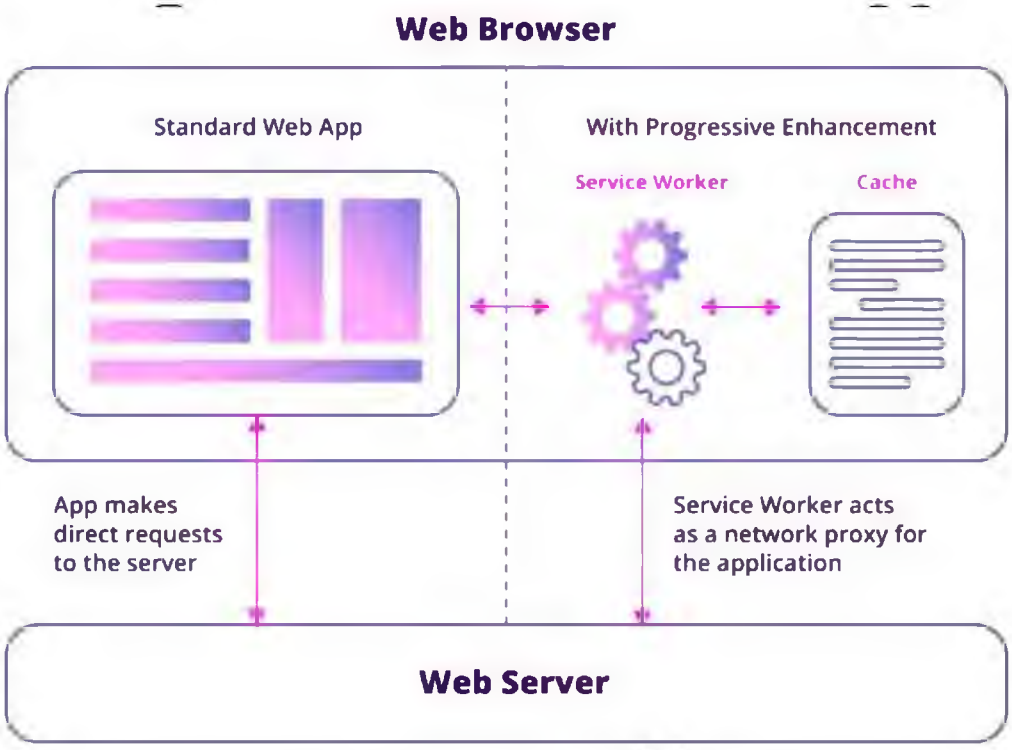


Рисунок 2.3 - Схема PWA

Реалізація PWA дозволяє веб-програмі підтримувати роботу в автономному режимі, фонову синхронізацію та push-повідомлення. Це відкриває доступ до функцій, які раніше вимагали рідної програми. У той же час, вибираючи архітектуру PWA для свого проекту, потрібно мати на увазі, що більшість функцій недоступні на iOS. Варто проаналізувати кожен конкретний бізнес-кейс.

Прогресивна архітектура веб-додатків підтримується Windows, Android та iOS (але для iOS автономний режим вимкнено). Розробники можуть додавати оновлення до веб-програми віддалено. PWA безпечний, оскільки використовує HTTPS. У той же час кінцеві користувачі можуть встановити PWA, навіть не відвідуючи Play Market або App Store. Серед недоліків цього типу архітектури – необхідність вибору браузера та ОС, які повністю його підтримують.

Обрані архітектурні рішення

Щоб веб-додаток забезпечував максимальну продуктивність, при його

розробці слід враховувати деякі нюанси. Веб-додаток має відповідати наступним умовам:

- Стабільна робота, відсутність аварій і помилок;
- Можливість простої масштабованості;
- Простий у використанні кінцевим користувачем;
- Швидке реагування на дії користувача;
- Відсутність точок зупинки;
- Використовувати новітні стандарти та технології;
- Безпека даних користувача.

Оскільки програми виявлення захворювань можуть бути комерціалізовані в майбутньому, дуже важливо негайно встановити обов'язкову реєстрацію користувача для використання функцій сайту. Це дозволить вам легко додати етап вибору підписки до процесу реєстрації при зміні моделі монетизації та негайно обмежити ролі користувачів. У цьому випадку, щоб пришвидшити роботу сайту та зменшити кількість статичних сторінок, а також зайвих HTTP-запитів до сервера, рекомендується використовувати принципи архітектури SPA – реєстрація та авторизація користувача буде завершуватися pop -ups-не потрібно вводити нову сторінку. Функція програми буде закрита до авторизації користувача, після впровадження на сайті з'явиться спеціальний блок, де ви зможете покроково вибирати рослини, завантажувати фотографії та аналізувати їх, а також отримувати результати нейронної мережі. та порівняння Рекомендації щодо подальших дій та обробки рослин.

Після аналізу всіх варіантів організаційної структури та планування функцій веб-додатку для виявлення хвороб рослин було прийнято таке рішення: додаток буде побудовано на принципах архітектури SPA.

Таким чином, будуть реалізовані основні вимоги сучасної архітектури веб-додатків.

– Зручний і функціонально орієнтований інтерфейс полегшить кінцевим користувачам використання програми;
– Швидкість веб-сайту забезпечуватиметься моделлю SPA – усі дані будуть завантажені з сервера одночасно, і не буде зайвих завантажень та очікувань під час використання програми.

2.3 Вибір мови програмування

Оскільки програма ідентифікації хвороб рослин працюватиме як веб-додаток, мовою програмування для дипломного проекту було обрано

JavaScript підтримує об'єктно-орієнтовані, функціональні та функціональні стилі. Це реалізація стандарту ECMAScript. Він найбільш широко використовується у браузерах як інструмент сценаріїв для взаємодії з веб-сторінками, але також використовується для розробки серверів, комп’ютерного програмного забезпечення та мобільних додатків. Наразі важко знайти напрям, у якому не можна було б рухатись, використовуючи JavaScript – серед суспільства розробників на GitHub він займає першу стрічку у рейтингу популярності мов програмування.

На JavaScript вплинули багато мов, але ця мова не належить жодній організації, що відрізняє її від багатьох інших альтернатив, що використовуються у веб-розробці. Як і HTML і CSS, JavaScript є однією з основних технологій у всесвітній мережі. JavaScript включає інтерактивні веб-сторінки і є невід'ємною частиною будь-якого веб-додатка. Більшість веб-сайтів використовують його для організації поведінки на стороні клієнта, і всі основні веб-браузери мають спеціальний механізм JavaScript для його виконання. Як мова багатьох парадигм, JavaScript підтримує функціональні та імперативні стилі програмування. Він має інтерфейс прикладного

програмування (API) для обробки тексту, дат, регулярних виразів, стандартних структур даних і об’єктної моделі документа (DOM). Однак сама мова не містить жодного вводу-виводу, наприклад мережеві інструменти, репозиторії чи графічні інструменти, оскільки середовище хоста (зазвичай веб-браузер) надає ці API. Двигуни JavaScript спочатку використовувались лише у веб-браузерах, але тепер вони вбудовані в деякі сервери, як правило, через Node.js.

2.4 Вибір бібліотеки для машинного навчання

Незважаючи на те, що Javascript є третьою найпопулярнішою мовою програмування в Github для машинного навчання, більшість науковців та компаній віддають перевагу Python як обраній мові. У Javascript Machine Learning спостерігався стрибок зростання у 2018 році, коли з’явилося декілька потужних інструментів для машинного навчання з Javascript.

Synaptic — це добре відома бібліотека нейронної мережі JavaScript, створена MIT, яку можна використовувати з Node.js або браузером. Однією важливою особливістю цієї бібліотеки є її здатність створювати та тренувати будь-яку архітектуру нейронної мережі першого чи другого порядку завдяки безархітектурному алгоритму та попередньо виготовленій структурі.

Вона також може імпортувати або експортувати мережі в JSON як окрему функцію, щоб вони могли підключатися до інших мереж або навіть плюзних з’єднань. Synaptic включає в себе деякі цікаві вбудовані архітектури, такі як мережі Хопфілда, кінцеві автомати, багатошарові персептрони, мережі довгострокової пам'яті (LSTM) тощо. І оскільки Synaptic є бібліотекою з відкритим вихідним кодом, будь-хто може зробити внесок у неї або використовувати її безкоштовно.

Synaptic також включає в себе тренажер, який здатний тренувати будь-

яку конкретну нейронну мережу за допомогою таких тестів, як тест Embedded Reber Grammar, розв'язування XOR, виконання завдання відволікання послідовності та вбудованих навчальних завдань. Це також допомагає в порівнянні продуктивності різних архітектур нейронних мереж.

Brain.js — це швидкодіюча бібліотека на основі JavaScript, яка використовується для машинного навчання та нейронних мереж. Його можна використовувати в браузері або з Node.js. З Brain.JS для різних завдань доступні різні типи мереж. Вона забезпечує підтримку різних нейронних мереж , такі як Long короткочасної пам'яті M.M. , рецидивний M.M. , і

Brain.js — це бібліотека швидкої обробки завдяки використанню графічного процесора для обчислень. Навіть якщо графічний процесор недоступний, він повертається до чистого JS і продовжує обробку. Brain.js надає кілька реалізацій нейронних мереж і заохочує створювати навчання та запускати ці нейронні мережі на стороні сервера разом з Node.js.

Ще одна перевага цієї бібліотеки полягає в тому, що для роботи з нею не потрібно бути чітко обізнаним з нейронними мережами. Щоб інтегрувати свій веб-сайт з цими мережевими моделями, вам просто потрібно реалізувати їх як функцію або використовувати формат JSON.

Brain.js можна використовувати для швидкого створення простої нейронної мережі за допомогою мови високого рівня. Це дозволяє створювати дійсно цікаву функціональність лише за допомогою кількох рядків коду та хорошого набору даних. Крім того, Brain.JS надає можливість працювати на стороні клієнта JavaScript.

Mind, написана на JavaScript, — це абсолютно гнучка бібліотека для нейронних мереж і роботи з браузерами та Node.js, щоб робити кращі прогнози. Однією з ключових особливостей Mind є те, що він обробляє навчальні дані, використовуючи матричну реалізацію, дозволяючи

розробникам налаштовувати топологію мережі.

Почати роботу з цією бібліотекою дуже зручно, оскільки вона швидко підключається та легше завантажувати та завантажувати плагіни, ніж інші бібліотеки. Ще одним плюсом Mind є простота налаштування попередньо навчених мереж.

ConvNetJS – це бібліотека JavaScript, яка спеціально розроблена для навчання моделей глибокого навчання та роботи з нейронними мережами. Найважливішою особливістю цієї бібліотеки є те, що вона повністю залежить від браузерів, тому будь-яке інше спеціальне програмне забезпечення, наприклад GPU, компілятори взагалі не потрібні. ConvNetJS також підтримує

ConvNetJS складається із звичайних нейронних мережевих модулів, які мають повністю пов'язані шари та нелінійності. Ця бібліотека має можливість формулювати й розв'язувати нейронні мережі за допомогою простого JavaScript, одночасно пропонуючи підтримку деяких поширених мережевих модулів.

Він також пропонує функції для визначення нейронних мереж, класифікації та помилок, згорткові мережі для обробки зображень, експериментальний модуль навчання з підкріпленням, заснований на Deep Q Learning, і додатковий модуль навчання, який все ще знаходиться на експериментальному рівні.

ML5.js — це повноцінна повна бібліотека з відкритим вихідним кодом для машинного навчання за допомогою Node.js і браузерів. Ви можете додати власні залежності, використовуючи його з Node.js.

Він побудований на основі TensorFlow і не має жодних зовнішніх залежностей. Подібно до Tensorflow, ця бібліотека може обробляти математичні операції, які прискорюються GPU, крім керування пам'яттю для алгоритмів машинного навчання.

ML5.js дозволяє легко отримати доступ до багатьох попередньо навчених алгоритмів машинного навчання у браузері, щоб його можна було використовувати для різних цілей, таких як виявлення мови тіла та висоти, налаштування зображень, створення текстів, пошук мовних зв'язків англійською мовою, створення музичні треки тощо.

Ця бібліотека має повноваження забезпечити інтенсивне розуміння машинного навчання разом із різними складностями, такими як етичні обчислення та збір даних, що робить її придатною навіть для новачка.

ML5.js пропонує утиліти для неконтрольованих і контрольованих проблем, а також критично важливих і простих моделей. Крім того, це універсальна бібліотека машинного навчання JavaScript загального призначення для розробників Typescript і JavaScript, яка включає допоміжні бібліотеки для математики та статистики, алгоритмів регресії, штучних нейронних мереж, навчання без нагляду та контролю, вилучення функцій, лінійних моделей, пакетування. , ансамбль, декомпозиція, кластеризація тощо.

Мало того, ML5.js також дозволяє генерувати випадкові числа, сортувати, виконувати бітові операції з масивами та хеш-таблицями — і навіть надає користувачам рутину для оптимізації, маніпулювання масивами та лінійної алгебри. Ще однією великою перевагою цієї бібліотеки є її підтримка перехресної перевірки.

Neuro.js — це фреймворк JavaScript для розробки та навчання моделей навчання з підкріпленням та моделей глибокого навчання, які широко використовуються у створенні помічників із технологіями AI та чат-ботами.

Багато розробників використовують цю бібліотеку для розробки, практики та навчання моделей глибокого навчання та машинного навчання, а потім розгортають їх у веб-переглядачі або на Node.js із його сценаріями JS.

Деякі з основних переваг цієї бібліотеки полягають у тому, що вона допомагає брати участь у класифікації в реальному часі, надає онлайн-підтримку для навчання та підтримує класифікацію форм із кількома мітками

під час створення проектів ML. Подивіться наведений нижче приклад коду класифікації кольорів, створений за допомогою бібліотеки Neuro.js.

Keras.js можна вважати другим за поширеністю фреймворком JS для глибокого навчання після TensorFlow.js. Він дуже популярний серед розробників, які працюють з бібліотеками нейронних мереж. Оскільки декілька фреймворків використовуються Keras для бекенда, ви можете навчати моделі в CNTK, TensorFlow та інших фреймворках.

Моделі машинного навчання, створені за допомогою Keras, можна запускати у браузері. Хоча моделі також можна запускати в Node.js, для цього буде доступний лише режим ЦП. Прискорення GPU не буде.

Багато провідних компаній, таких як Netflix і Uber, працюють з моделями нейронних мереж Keras, щоб покращити роботу користувачів. Ряд наукових організацій, таких як NASA , CERN та ін., використовують цю технологію для своїх проектів, пов'язаних зі штучним інтелектом. Keras вважається альтернативою JS для бібліотеки штучного інтелекту, і він дозволяє вам виконувати різні моделі у своєму проекті та використовувати підтримку графічного процесора, яка надається API 3D-дизайну WebGL.

TensorFlow.js – це бібліотека JavaScript з відкритим кодом, створена Google Brain gathering. Він сприяє апаратному прискоренню завдяки повному та гнучкому набору інструментів. Завдяки своїм рівням глибокого навчання та всеосяжній лінійній алгебрі ця бібліотека стала хлібом і маслом для всіх проектів JavaScript, заснованих на машинному навчанні.

TensorFlow.js дозволяє користувачам навчати нейронні мережі за допомогою браузера або виконувати попередньо навчені моделі в режимі висновку, одночасно вносячи будівельні блоки машинного навчання в Інтернет. Ви можете запустити моделі TensorFlow за замовчуванням, які зараз доступні, або навіть конвертувати їх у деякі моделі Python як доповнення.

TensorFlow.js також включає деякі вже існуючі моделі машинного

навчання. Їх можна використовувати для перенавчання ваших власних даних. Він також надає можливість розгортати моделі машинного навчання в будь-якому місці, включаючи пристрій, незалежно від мови, яку ви використовуєте, локально, у браузері чи хмарі.

Попри всі сильні сторони названих вище бібліотек, для виконання цього дипломного проекту було обрано Tensorflow.js. У 2020 ця бібліотека стала головним вибором для всіх серйозних проектів JavaScript з використанням машинного навчання. Tensorflow швидко перевершив свою версію Python за кількістю підтримуваних API. На даний момент майже всі проблеми машинного навчання можна вирішити за допомогою цієї бібліотеки. Окрім забезпечення глибокого навчання та машинного навчання в середовищі Node.js, Tensorflow.js також можна використовувати безпосередньо у браузері для прискорення за допомогою WebGL. Оскільки веб-додаток для виявлення хвороб рослин буде запускатися в браузері, цей факт є одним з вирішальних факторів при виборі цієї бібліотеки – Tensorflow забезпечить швидкість роботи програми не тільки при її використанні кінцевим користувачем, а й при Розробка та навчання моделі TensorFlow також включає TensorBoard, який є інструментом візуалізації браузера для оцінки ефективності навчання та мережевих параметрів моделі. TensorFlow досягає своєї продуктивності за рахунок розпаралелювання завдань між центральним і графічним процесором. Ядро кожної операції реалізовано на C ++ із застосуванням бібліотек Eigen і cuDNN для кращої продуктивності.

2.5 Вибір програмних засобів

У процесі розробки додатку головними технологіями були: на клієнтській частині React, HTML, CSS, а на серверній частині — Node.js, Express MongoDB. Інформація про технології Javascript та Tensorflow вже була

представлена в даній роботі, тому далі більше детально буде обґрунтовано вибір інших технологій, які використовувались для реалізації проекту.

Мова розмітки гіпертексту (Hypertext Markup Language), або, як його частіше називають, HTML, - це комп'ютерна мова, що лежить в основі World Wide Web (Всесвітньої павутини). Завдяки мові HTML будь-який текст можна розмітити, перетворивши його на гіпертекст з подальшою публікацією на Web.

Мова HTML має власний набір символів, за допомогою яких веб-браузери відображають сторінку. Ці символи, звані дескрипторами, включають елементи, необхідні для створення гіперпосилань

Однією з відмінних рис HTML-документів є те, що сам документ містить тільки текст, а всі інші об'єкти вбудовуються в документ в момент його відображення Браузером за допомогою спеціальних тегів і зберігаються окремо. При збереженні HTML-файлу в місці розміщення документа створюється папка, в яку розміщуються супутні йому графічні елементи оформлення

Каскадні таблиці стилів або просто таблиці стилів (CSS - Cascading Style Sheets) - це набір правил, що описують форматування різних фрагментів відображає форматування одного фрагмента або однотипної групи фрагментів коду, називається стилем. Таблиці стилів описуються особливою мовою CSS і зберігаються в спеціальних файлах з розширенням CSS, хоча можуть бути впроваджені в саму Web-сторінку.

React — це інструмент для створення як компонентів інтерфейсу користувача, так і цілих інтерфейсів — усього, що стосується об’єднання візуальних елементів, прив’язки даних до цих елементів і визначення логіки, що керує цим.

React.js можна використовувати для створення інтерфейсів користувача

на JavaScript для різних платформ. Ви можете використовувати ReactDOM для веб-додатків, React Native для розробки мобільних додатків (спільне використання більшості коду між Android та iOS) і кросплатформні гібридні настільні програми з Electron. Нещодавно Microsoft також випустила React Native для Windows.

Функція віртуальної DOM у React.js підвищує швидкість веб-додатків. Він створює список усіх компонентів з його функціями та вмістом як властивості та об'єкти.

Render() React — це те, що створює дерево вузлів із цих елементів реакції. Пізніше він оновлює дерево як відповідь на ті мутації, знайдені в структурі даних. Зазвичай це відбувається через різні дії, які виконуються в інтерфейсі користувача.

Nodejs (або просто Node) - це серверна платформа для роботи з JavaScript через двигун V8. JavaScript виконує дію на стороні клієнта, а Node – на сервері. За допомогою Node можна писати повноцінні програми. Node вміє працювати із зовнішніми бібліотеками, викликати команди з коду на JavaScript та виконувати роль веб-сервера.

Переваги Node над іншими серверними технологіями (Java, PHP, C#) дуже прості: з Node простіше масштабуватися. При одночасному підключенні до сервера тисяч користувачів Node працює асинхронно, тобто ставить пріоритети та розподіляє ресурси грамотніше. Java ж, наприклад, виділяє на кожне підключення окремий потік.

Nodejs Express є веб-фреймворком, написаним на JavaScript і працюючим усередині середовища виконання node.js. Цей модуль висвітлює деякі ключові переваги цього фреймворку, встановлення середовища розробки та виконання основних завдань веб-розроблення та розгортання. Деякі переваги даного фреймворку:

Дозволяє настроїти посередників для відповіді на запити HTTP.

- Визначає таблицю маршрутизації, яка використовується для виконання різних дій на основі методу HTTP та URL-адреси.

- Дозволяє динамічно створювати HTML-сторінки на основі передачі аргументів шаблонам.

MongoDB — програма керування базами даних NoSQL з відкритим вихідним кодом . NoSQL використовується як альтернатива традиційним реляційним базам даних. Бази даних NoSQL досить корисні для роботи з великими наборами розподілених даних. MongoDB — це інструмент, який може керувати документально-орієнтованою інформацією, зберігати або отримувати інформацію.

MongoDB підтримує різні форми даних. Це одна з багатьох технологій нереляційних баз даних, які виникли в середині 2000-х під прапором NoSQL - зазвичай для використання в програмах для великих даних та інших роботах з обробки даних, які погано вписуються в жорстку реляційну модель. Замість використання таблиць і рядків, як у реляційних базах даних , архітектура MongoDB складається з колекцій і документів.

MongoDB використовує записи, які складаються з документів, які містять структуру даних, що складається з пар полів і значень. Документи є основною одиницею даних у MongoDB. Документи подібні до JavaScript Object Notation , але використовують варіант під назвою Binary JSON (BSON). Перевага використання BSON полягає в тому, що він вміщує більше типів даних. Поля в цих документах подібні до стовпців у реляційній базі даних. Містяться значення можуть бути різних типів даних, включаючи інші документи, масиви та масиви документів, відповідно до посібника користувача MongoDB. Документи також включатимуть первинний ключ як унікальний ідентифікатор.

Набори документів називаються колекціями , які функціонують як еквівалент таблиць реляційної бази даних. Колекції можуть містити будь-які типи даних, але обмеження полягає в тому, що дані в колекції не можуть бути

розподілені між різними базами даних.

Розробка блок-схеми алгоритму

Блок-схема — це спеціальна схема, яку зазвичай використовують інженери для візуалізації системи та її різних взаємодій. Блок-схеми ідеально підходять для створення оглядів системи високого рівня, а також для ілюстрації ключових компонентів системи, візуалізації вхідних і вихідних даних і розуміння робочих взаємовідносин у системі.

Часто алгоритм використовується не як інструкція для комп'ютера, а як схема виконання будь-яких дій. Це дозволяє відзначити дієвість даного методу вирішення, виправити можливі помилки і порівняти його з іншими подібними рішеннями ще до імплементації рішення. Крім того, алгоритм є основою для складання програми, яка повинна бути написана мовою програмування для подальшої реалізації процесу обробки інформації на ПК. На сьогоднішній день стали відомі два практичні способи побудови таких послідовностей. Перший — це покроковий словесний опис, а другий — блок-схема алгоритму задачі. Перший з них був значно менш поширений. Це пов’язано з недостатньою наочністю та багатослівністю. Другий спосіб, навпаки, є дуже зручним засобом відображення послідовності.

Нижче наведена блок-схема програми, що описує процес використання додатку користувачем та процес роботи програми, включаючи такі процеси:

- перевірка авторизації користувача;
- реєстрація та авторизація у додатку;
- роутинг певних частин додатку у залежності від статусу авторизації;
- завантаження на валідації даних для класифікації;
- використання моделі для розпізнавання хвороб;
- демонстрація результатів роботи нейронної мережі.

Рисунок 2.4 - Блок схема алгоритму

РОЗРОБКА СИСТЕМИ

Підготовка набору даних для навчання моделі

Успішне використання глибоких нейронних мереж для розпізнавання та класифікації зображень в основному залежить від збору правильних наборів даних. Якісний набір даних необхідний для досягнення максимально можливої точності. Існують певні правила для підготовки правильного набору даних для навчання моделі.

Для того, щоб мережа охопила характеристики кожного класу та розпізнала нові зображення, дуже важливо використовувати велику кількість різноманітних наборів даних для навчання.

Важливо:

- якісні зображення та фокус;
- знімки об'єктів дослідження з різних ракурсів;
- зображення у різних кольорах та під різним освітленням;
- об'єкти дослідження в різних формах і напрямках;
- найкраще мати фотографії з різним фоном;
- зображення з різними розмірами об'єктів і різними відстанями.

Для навчання та тестування використовувався набір даних PlantVillage відкритого доступу, який складається з 54 000 здорових і заражених листків рослин. Ця база даних містить 38 різних класів 14 різних видів рослин із зображеннями здорових і уражених хворобами листків. Всі зображення були зроблені в лабораторних умовах.

В усіх експериментах використовується три різні версії всього набору даних PlantVillage. Починаючи з основного набору даних PlantVillage у кольорі, було прийнято рішення провести певні експеременти з версією набору даних PlantVillage у чорно-білому представленні. Далі були використані ті ж самі набори зображень, але на них листя все були сегментовані - таким чином

всю додаткову зайву інформацію було видавлено з зображень. Цей набір експериментів був здійснений, щоб покращити результати розпізнавання та переконатись у тому, що нейронна мережа дійсно вивчає «поняття» хвороб рослин, а не орієнтується на зайві фонові ознаки. На малюнку показані різні версії одного і того ж листа для випадково вибраного набору листя.

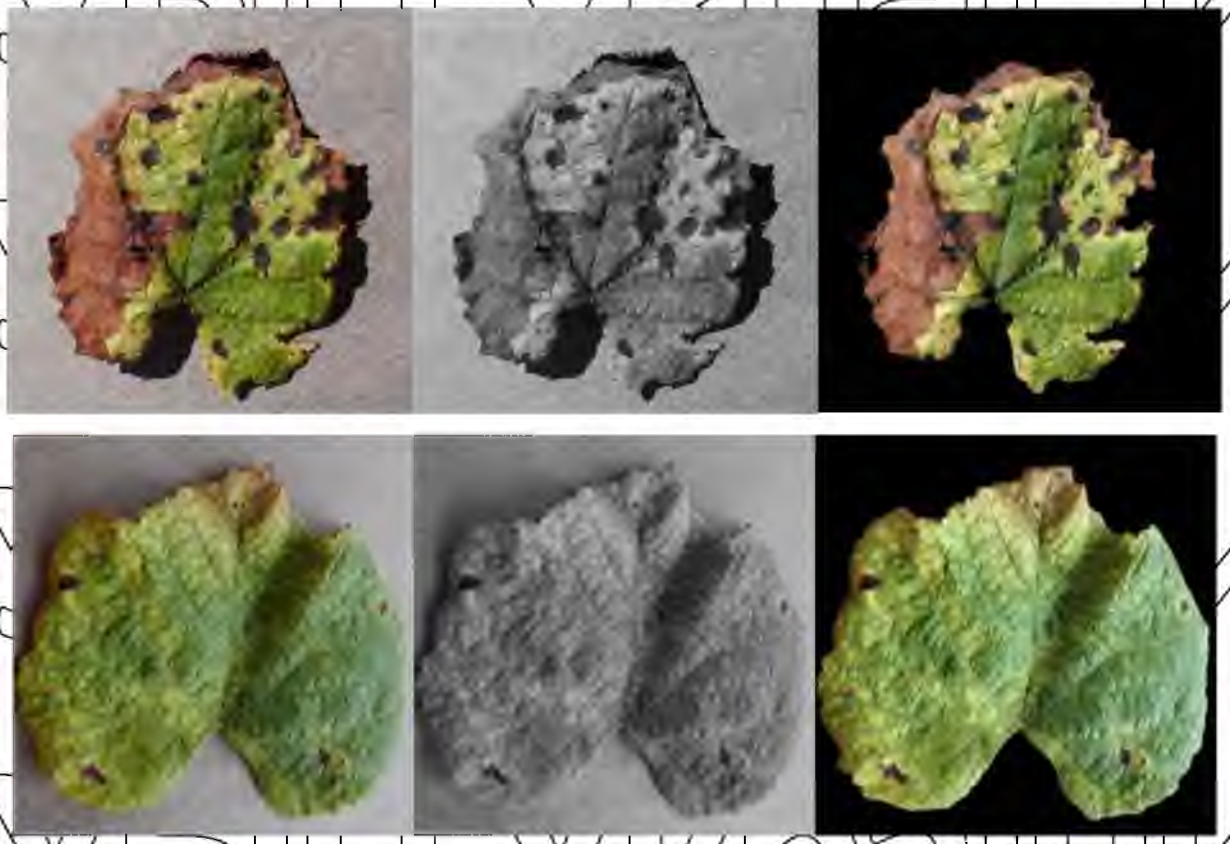


Рисунок 3.1 – Приклад зображень для тренування моделі

3.1.1 Розділення набору даних

Важливим кроком у машинному навчанні є поділ даних на різні набори для різних цілей. Найкраще рішення - створити три колекції:

– Навчальний набір: більшість ваших даних має бути в цьому навчальному наборі. Ці дані використовуються для вивчення моделі, для вивчення параметрів моделі, тобто ваги класифікатора нейронної мережі.

– Валідаційний набір: під час навчання також використовується окремий валідаційний набір (іноді його називають набором для

розробки). Хоча ваги нейронної мережі оновлюються на основі даних навчання, кожна нейронна мережа вимагає від користувачів вказати багато гіперпараметрів (наприклад, рівень навчання тощо). Вибір гіперпараметрів сильно впливає на процес навчання та отриману модель. Найкраще перевірити різні значення цих гіперпараметрів і оцінити їх вплив на результати навчання, вимірюючи продуктивність створеної мережі на цьому наборі перевірки.

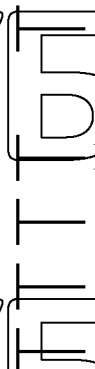
– Тестовий набір: набір окремих зображень, міток може бути відсутнім. Ці дані не використовуватимуться в жодній частині побудови моделі або процесу навчання. Ці зображення можуть використовуватись щоб оцінити ефективність остаточної моделі.

Після формування набору даних з фотографій необхідної якості та відповідності вищезазначеним критеріям їх слід помістити в категорії з відповідними класами. Рекомендується використовувати принаймні 100 навчальних зображень для кожного класу, щоб досягти розумної ефективності класифікації, але це може залежати від типу зображення у конкретному випадку використання. Враховуючи, що зображення листя хворих рослин дуже схожі один на одного, а відмінності між різними захворюваннями не завжди легко вловити, для кожної категорії підготовлено від 500 до 5000 фотографій.

Після підбору та розподілення навчальних даних була отримана наступна структура папок з зображеннями:

├── Вірус мозаїки
├── Септоріоз
├── Ранній фітофтороз
├── Здоровий

├── Вірус мозаїки
├── Септоріоз
├── …



Це набір даних для навчання нейронної мережі для розпізнавання хвороб помідорів. Аналогічні набори було створено для винограду, яблуні та картоплі.

3.2 Тренування нейронної мережі

3.2.1 Тренування згорткових мереж для класифікації зображень

Якщо зрівнювати з мережами прямого поширення, де кожен вхідний нейрон з'єднаний з вихідним нейроном наступного шару, згорткові мережі використовують згортку на кожному вхідному шарі для отримання результуючого значення. Операція згортки використовує вагову матрицю маленького розміру, яка рухається по всьому шару обробки, і після кожного руху генерує сигнал активації наступного шару нейронів з подібними положеннями. Ця матриця називається ядром згортки; вона використовується для різних нейронів вихідного шару.

Згортку можна пояснити на прикладі обробки зображень. Якщо ми думаємо про зображення як про двовимірну функцію, то різні перетворення зображення є не що інше, як згортка функції зображення та локальної функції - ядром згортки.

Ось наприклад одновимірний згортковий шар, де Xn — входи, aYn - виходи. Тоді наступна функція буд представляти виходи:

Стандартна матриця ваг з'єднує кожен вхід з кожним нейроном з різними вагами. Матриця для згорткового шару виглядає інакше через те, що різні ваги можуть з'являтися на кількох позиціях, а через те що нейрони св’язані з усіма можливими входами, матриця містить безліч нульових елементів.

Кожен новий піксель зображення являє собою зважену суму пікселів, які ядро пройшло до цього моменту часу.

У процесі виконання згортки зображення кожен фрагмент множиться на матрицю згортки, після чого результат сумується і записується в відповідну позицію вихідного зображення. Матрицею згортки — це графічне кодування пивної ознаки. Отриманий у результаті згортки наступний шар ілюструє наявність цієї ознаки. Згорткова нейронна мережа має багато наборів ваг, які кодують елементи зображень. Ядра згортки формуються в процесі навчання мережі. З кожним проходом наборів ваг формується карта ознак. Оскільки з'являється багато незалежних карт ознак на одному шарі, то мережа стає багатоканальної.

Кожен канал кожного згорткового шару має свій власний фільтр, і його ядро згортки обробляє попередній шар у вигляді фрагментів. Далі об’єднуються результати застосування різних фільтрів — і у результаті ми отримуємо шари об'єднання. Операція субдискретізація зменшує розмірність згенерованої карти ознак. У цій архітектурі мережі вважається, що інформація про існування необхідного об’єкта важливіша за точне розуміння його координат, тому береться максимальне значення з кількох сусідніх нейронів карти ознак вибирається максимальний і приймається за один нейрон ущільненої карти ознак меншої розмірності. Завдяки цій операції, крім прискорення подальших обчислень, масштаб вхідного зображення мережею стає незмінним.

Після початкового шару сигнал проходить через серію згорткових шарів, де чергуються операції згортки та об’єднання. Чергування шарів дозволяє створювати карти ознак: на кожному наступному шарі розмір карти буде

зменшуватися, а кількість каналів збільшуватися. На практиці це означає можливість ідентифікувати складні ієрархічні структури ознак.

Після кількох шарів карта об’єктів вироджується у вектор або скаляр, але таких карт ознак є сотні. На вихідному кінці мережевого згорткового шару додатково встановлюються кілька додаткових повних шарів нейронної мережі (наприклад, персептрони), а на вхідному кінці надається остаточна карта ознак.

3.2.2 Гіперпараметри мережі

Гіперпараметрами згорткової нейронної мережі є:

– вузька і широка згортки (wide and narrow convolutions): нульове заповнення дозволяє зробити згортку ширшою, наприклад, коли перший елемент матриці не має сусідніх елементів зліва та зверху. Не додаючи нулів, отримуємо вузьку згортку;

– розмір кроку(stride): розмір кроку визначає зміщення фільтра для кожного кроку. Чим більший розмір кроку, тим менший застосований фільтр і менший розмір вихідної матриці. Зазвичай використовується розмір кроку, що дорівнює одиниці, але більший розмір кроку дозволяє побудувати модель, яка веде себе як рекурентна нейронна мережа (тобто згортка з великим розміром кроку виглядає як дерево);

– шари об'єднання: шари об'єднання допомагає зменшити розмірність вихідної інформації, зберігаючи при цьому найбільш видиму інформацію;

– вентиляційні канали: канали – це різні «погляди» на вхідні дані. Наприклад, зазвичай є три канали розпізнавання зображень - RGB;

3.2.3 Навчання нейронних мереж

Загальні поняття в навчанні нейронних мереж наступні:

– епоха - прямий і зворотний прохід через весь набір навчальних прикладів;

– Розмір серії (пакет) – кількість навчальних прикладів для ітерації прямого та зворотного проходу;

– Кількість ітерацій – кількість проходів: використовуйте приклади (пакет) для кожного проходу. Один прохід = прямий прохід + зворотний прохід.

Іншими словами, з тисячою прикладів, а пакетом рівним 500, нам знадобиться дві ітерації, щоб завершити епоху.

При навчанні нейронних мереж дуже поширений алгоритм зворотного поширення помилки. Алгоритм зворотного поширення помилки використовує метод градієнтної оптимізації для визначення стратегії вибору ваги багатошарової мережі. Оскільки цільова функція (зазвичай визначається як квадратична різниця суми фактичного початкового значення та очікуваного початкового значення) є безперервною, метод градієнтної оптимізації є дуже ефективний у навчанні мережі. При навчанні багатошарової нейронної мережі необхідно розрахувати вектор градієнта щодо параметрів кожного шару мережі.

Основна складність навчання нейронних мереж полягає в методі подолання локального мінімуму. Недоліком градієнтного спуску при навчанні мережі є параліч мережі. Це відбувається, коли значення ваги мережі в результаті корекції стає дуже великим. Оскільки помилка, яка повертається під час процесу навчання, пропорційна похідній функції стиснення, процес навчання може майже зупинитися. Цьому можна запобігти, зменшивши розмір кроку η, але процес навчання займе більше часу.

Іншою проблемою є низький розмір кроку. Якщо значення кроку не змінюється і є досить малим, швидкість навчання занадто повільна. Якщо розмір кроку занадто великий, мережа може бути паралізована. Потрібно змінити значення кроку: збільшувати до тих пір, поки не припиниться покращення оцінки в напрямок антіградієнта, та навпаки зменшувати - якщо оцінка не покращується.

3.2.4 Трансферне навчання

Складні моделі глибокого навчання мають мільйони параметрів (ваг), і навчання їх з нуля часто вимагає великих обсягів даних обчислювальних

ресурсів. Трансферне навчання - це метод, який скорочує більшу частину цього процесу, беручи частину моделі, яка вже була навчена відповідній задачі, і повторно використовуючи її в новій моделі.

Цей підхід дозволяє побудувати класифікатор зображень, який використовує переваги моделі, яка вже була навчена розпізнавати 1000 різних типів об'єктів у зображеннях. За допомогою трансферного навчання, використовується набагато менше навчальних даних, ніж потрібно для вихідної моделі. Це корисно для швидкої розробки нових моделей, а також для налаштування моделей в ресурсномістких середовищах, таких як браузери і мобільні пристрої.

Оскільки у рамках даного проекту розробляється веб-додаток, який може використовуватися на слабких комп'ютерах або мобільних пристроях, оптимальним рішення буде використання трансферного навчання нейронної мережі. Tensorflow дає подібні можливості. Ця бібліотека дозволяє не тільки навчати глибокі нейронні мережі, але й утримує кілька готових попередньо навчених мереж. Ці мережі можна використовувати, не витрачаючи значний час та обчислювальні ресурси на навчання. Для розпізнавання об'єктів на зображеннях з набору даних буде використовуватись мережа MobileNet (проект зі створення та супроводу масивної бази даних анотованих зображень, призначена для відпрацювання та тестування методів розпізнавання образів та машинного зору).

У процесі тренування мережі були використані різні набори параметрів, щоб потім можна було порівняти отримані результати. Більше детальної інформації про результати експерементів буде у останньому розділі цієї роботи.

Лістинг 3.1 - Підключення бібліотеки Tensorflow

```
<script
src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@1.3.1/dist/tf
.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
```

```
<script src="https://cdn.jsdelivr.net/npm/@tensorflow-
models/mobilenet"></script>
```

Лістинг 3.2 – Код підготовки зображень для тренування

```
export class GetData {
  constructor() {}
  async fetchData() {
    const image = new Image();
    const canvasContainer = document.createElement("canvas");
    const context = canvasContainer.getContext("2d");
    const fetchImage = new Promise((resolve, reject) => {
      image.crossOrigin = "";
      image.onload = () => {
        image.width = image.naturalWidth;
        image.height = image.naturalHeight;
        const datasetBuffer = new ArrayBuffer(
          NUM_DATASET_ELEMENTS * IMG_SIZE * 4
        );
        const batchSize = 16;
        canvasContainer.width = image.width;
        canvasContainer.height = batchSize;
        for (let i = 0; i < NUM_DATASET_ELEMENTS / batchSize;
i++) {
          const datasetBytes = new Float32Array(
            datasetBuffer,
            i * IMG_SIZE * batchSize * 4,
            IMG_SIZE * batchSize
          );
          context.drawImage(
            image,
            0,
            i * batchSize,
            image.width,
            batchSize,
            0,
            0,
            image.width,
            batchSize
          );
          const imageData = context.getImageData(
            0,
            0,
            canvasContainer.width,
            canvasContainer.height
          );
          for (let j = 0; j < imageData.data.length / 4; j++) {
            datasetBytes[j] = imageData.data[j * 4] / 255;
          }
        }
        this.datasetImages = new Float32Array(datasetBuffer);
        resolve();
```

```
      };
      image.src = IMAGES_PATH;
    });
    const labelsResult = fetch(LABELS_PATH);
    const [pictures, labels] = await Promise.all([fetchImage,
labelsResult]);
    this.trainingPackLabels = new Uint8Array(await
labels.arrayBuffer());
    this.trainingPictures = this.datasetImages.slice(
      0,
      IMG_SIZE * TOTAL_TRAIN_ELEMENTS
    );
    this.testPictures = this.datasetImages.slice(
      IMG_SIZE * TOTAL_TRAIN_ELEMENTS
    );
    this.trainLabels = this.trainingPackLabels.slice(
      0,
      TOTAL_CLASSES * TOTAL_TRAIN_ELEMENTS
    );
    this.testLabels = this.trainingPackLabels.slice(
      TOTAL_CLASSES * TOTAL_TRAIN_ELEMENTS
    );
  }
  getTrainingData() {
    const xs = tf.tensor4d(this.trainingPictures, [
      this.trainingPictures.length / IMG_SIZE,
      IMG_H,
      IMG_W,
      1,
    ]);
    const classes = tf.tensor2d(this.trainLabels, [
      this.trainLabels.length / TOTAL_CLASSES,
      TOTAL_CLASSES,
    ]);
    return {
      xs,
      classes
    };
  }
  getTestingData(amountOfExamples) {
    let xs = tf.tensor4d(this.testPictures, [
      this.testPictures.length / IMG_SIZE,
      IMG_H,
      IMG_W,
      1,
    ]);
    let classes = tf.tensor2d(this.testLabels, [
      this.testLabels.length / TOTAL_CLASSES,
      TOTAL_CLASSES,
    ]);
    if (amountOfExamples != null) {
      xs = xs.slice([0, 0, 0, 0], [amountOfExamples, IMG_H,
IMG_W, 1]);
```

```
      classes = classes.slice([0, 0], [amountOfExamples,
TOTAL_CLASSES]);
    }
    return {
      xs,
      classes
    };
  }
}
```

Функція тренування згорткової нейронної мережі, що використовує вбудовані функції та інтерфейси бібліотеки Tensorflow, наведена далі:

Листинг 3.3 – Код функції тренування

```
async function train(model, onIteration) {
  ui.logStatus("Training...");
  model.compile({
    optimizer,
    loss: "categoricalCrossentropy",
    metrics: ["accuracy"],
  });
  const batchSize = 64;
  const validationSplit = 0.2;
  const trainEpochs = 50;
  let trainBatchCount = 0;
  const trainData = data.getTrainData();
  const testData = data.getTestData();
  const totalNumBatches =
    Math.ceil((trainData.xs.shape[0] * (1 -
validationSplit)) / batchSize) *
    trainEpochs;
  let valAcc;
  await model.fit(trainData.xs, trainData.labels, {
    batchSize,
    validationSplit,
    epochs: trainEpochs,
    callbacks: {
      onBatchEnd: async (batch, logs) => {
        trainBatchCount++;
        ui.logStatus(
          `Training... (` +
          `${((trainBatchCount / totalNumBatches) *
100).toFixed(1)}%` +
          ` complete).`
        );
        ui.plotLoss(trainBatchCount, logs.loss, "train");
        ui.plotAccuracy(trainBatchCount, logs.acc, "train");
        if (onIteration && batch % 10 === 0) {
          onIteration("onBatchEnd", batch, logs);
```

```
            }
            await tf.nextFrame();
          },
          onEpochEnd: async (epoch, logs) => {
            valAcc = logs.val_acc;
            ui.plotLoss(trainBatchCount, logs.val_loss,
"validation");
            ui.plotAccuracy(trainBatchCount, logs.val_acc,
"validation");
            if (onIteration) {
              onIteration("onEpochEnd", epoch, logs);
            }
            await tf.nextFrame();
          },
        },
      });
      const testResult = model.evaluate(testData.xs,
testData.labels);
      const testAccPercent = testResult[1].dataSync()[0] * 100;
      const finalValAccPercent = valAcc * 100;
      ui.logStatus(
        `Final validation accuracy:
${finalValAccPercent.toFixed(1)}%; ` +
        `Final test accuracy: ${testAccPercent.toFixed(1)}%`
      );
    }
```

Розробка веб-додатку

Розробка додатку почалася зі створення серверної частини. Спочатку було добавлено головний стартовий файл сервера, у якому ініціалізується сервер, налаштовується під'єднання до база даних та виконуються налаштування роутера.

Лістинг 3.4 – Код головної функції сервера

```
const express = require("express");
const mongoose = require("mongoose");
const app = express();
const port = 9000;
const url =
"mongodb+srv://admin:admin1234@cluster0.l9cpj.mongodb.net/myFirs
```

```
tDatabase?retryWrites=true&w=majority";
mongoose.connect(url, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  createIndexes: true,
});
const con = mongoose.connection;
app.use(express.json());
try {
  con.on("open", () => {
    console.log("connected");
  });
} catch (error) {
  console.log("Error: " + error);
}
app.use("/api/auth", require("./routes/auth.routes"));
app.listen(port, () => {
  console.log("Server has been started");
});
```

Далі створено контролери для реєстрації та авторизації користувачів з логікою для цих операцій.

Лістинг 3.5 – Код контролерів реєстрації та авторизації

```
const {
  Router
} = require("express");
const bcrypt = require("bcryptjs");
const config = require("config");
const jwt = require("jsonwebtoken");
const {
  check,
  validationResult
} = require("express-validator");
const User = require("../models/User");
const router = Router();
router.post(
"/register",
[
  check("email", "Некорректный email").isEmail(),
  check("password", "Минимальная длина пароля 6
символов").isLength({
    min: 6,
  }),
],
async (req, res) => {
  try {
    const errorsArray = validationResult(req);
    if (!errorsArray.isEmpty()) {
```

```
            return res.stArrayatus(400).json({
              errors: errorsArray.array(),
              message: "Ошибка регистрации, некорректные
данные",
            });
          }
          const {
            email,
            password,
            lastName,
            firstName
          } = req.body;
          const maybeUser = await User.findOne({
            email
          });
          if (maybeUser) {
            return res.status(400).json({
              message: "Юзер уже есть"
            });
          }
          const hashedPswrd = await bcrypt.hash(password, 12);
          const user = new User({
            email,
            password: hashedPswrd,
            lastName,
            firstName,
          });
          await user.save();
          res.status(201).json({
            message: "Пользователь зарегестрирован!"
          });
        } catch (e) {
          res.status(500).json({
            message: "Серверная ошибка"
          });
        }
      }
    );
    router.post(
      "/login",
      [
        check("email", "Некорректный
email").normalizeEmail().isEmail(),
        check("password", "Пароль").exists(),
      ],
      async (req, res) => {
        try {
          const errorsArray = validationResult(req);
          if (!errorsArray.isEmpty()) {
            return res.status(400).json({
              errors: errorsArray.array(),
              message: "Ошибка логина, некорректные данные",
            });
```

```
        }
        const {
          email,
          password
        } = req.body;
        const user = await User.findOne({
          email
        });
        if (!user) {
          return res.status(400).json({
            message: "Пользователя нет"
          });
        }
        const isFound = await bcrypt.compare(password,
user.password);
        if (!isFound) {
          return res.status(400).json({
            message: "Неправильный пароль"
          });
        }
        const token = jwt.sign({
          userId: user.id
        }, config.get("jwtSecret"), {
          expiresIn: "1h",
        });
        res.json({
          token,
          userId: user.id,
          name: user.firstName + " " + user.lastName,
          email: user.email,
        });
      } catch (e) {
        res.status(500).json({
          message: "Серверная ошибка"
        });
      }
    }
  );
  module.exports = router;
```

При створенні клієнтської частини додатку використовувалася бібліотека React, тому фронтенд частина додатка була поділена на компоненти. Було створено декілька головних контейнерів — RegistationPage, LoginPage,

Для зареєстрованих користувачів відривається доступ до головного функціоналу веб-додатку. Після обрання класу рослини він може додати нове

зображення для класифікації.

Лістинг 3.6 – Код обробки завантаженого зображення

```
const [selectedFile, setSelectedFile] = React.useState();
const [preview, setPreview] = React.useState();
React.useEffect(() => {
if (!selectedFile) {
  setPreview(undefined);
  return;
}
  const objectUrl = URL.createObjectURL(selectedFile);
  setPreview(objectUrl);
  // eslint-disable-next-line consistent-return
  return () => URL.revokeObjectURL(objectUrl);
}, [selectedFile]);
const onSelectFile = (e) => {
if (!e.target.files || e.target.files.length === 0) {
  setSelectedFile(undefined);
  return;
}
  setSelectedFile(e.target.files[0]);
};
```

Коли дані для аналізу отримані та перевірені, вони надсилаються до відповідної моделі нейронної мережі для аналізу. Створено 4 незалежні моделі для кожної рослини - це не тільки дає змогу швидше навчати такі моделі, але й дозволяє легко розширювати програму, легко розширювати її функції та додавати інші моделі для аналізу, що відповідає важливому стандарту масштабованість архітектури сайту.

Лістинг 3.7 – Імпортування моделі та функція виклику передбачення

```
async function prepairModel() {
  modelGrapes = await tmImage.load("grapes/model.json",
    "grapes/metadata.json");
  modelApples = await tmImage.load("apples/model.json",
    "apples/metadata.json");
}
async function makePrediction() {
  const prediction = await model.predict(inputImage, false);
  setResult(prediction);
  setIsLoader(true);
  getDiseaseInfo(prediction);
}
```

ДОСЛІДЖЕННЯ ТА ТЕСТУВАННЯ СИСТЕМИ

Тестування веб-додатку

Ручне тестування програмного забезпечення – це процес перевірки програмного забезпечення, що виконується вручну. Це означає, що для цього не використовуються спеціальні автоматизовані засоби.

Програмне забезпечення тестують інженери-тестувальники, які грають роль кінцевих користувачів, моделюють ситуації на основі тестових сценаріїв і записують результати. Завдання ручного тестування програмного забезпечення — виявити будь-яку поведінку, яка відрізняється від очікуваної користувача. Це важливий крок у забезпеченні якості програмного забезпечення, метою якого є досконале вивчення програмного коду та виявлення помилок у роботі системи. Насправді, оскільки ручне тестування веб-додатків вимагає ручного тестування програмного продукту, це забезпечує точність кінцевого продукту. Тестувальники шукають проблеми, які можуть вплинути на здатність кінцевого користувача використовувати програму. Підприємства використовують ручне тестування як процес перевірки. Таким же чином компанії намагаються оцінити та кваліфікувати продукт веб-додатка за такими параметрами, як точність, повнота, зручність використання, ефективність тощо.

Ручне тестування ефективно долає обмеження, які несе з собою автоматичне тестування. У результаті команди розробників додатків, які працюють над користувацьким досвідом і графічними інтерфейсами з постійними оновленнями, для успіху в значній мірі покладаються на ручне тестування.

Основні переваги ручного тестування:

- тестовий проект можна швидко запустити;
- ручне тестування дозволяє при необхідності вносити корективи (змінювати та оновлювати тести);

– легка адаптація до динамічних змін системи;

– ручне тестування повністю імітує фактичне використання системи кінцевим користувачем;

– дозволяє отримувати індивідуальний відгук, особливо для одноразових і короткострокових проектів.

4.1.1 Користувачі без авторизації

Якщо користувачі переходять на сайт додатку без авторизації, вони не зможуть в повній мірі користуватися сервісом. При переході по будь-якому адресу користувач буде направлений на сторінку входу або реєстрації.

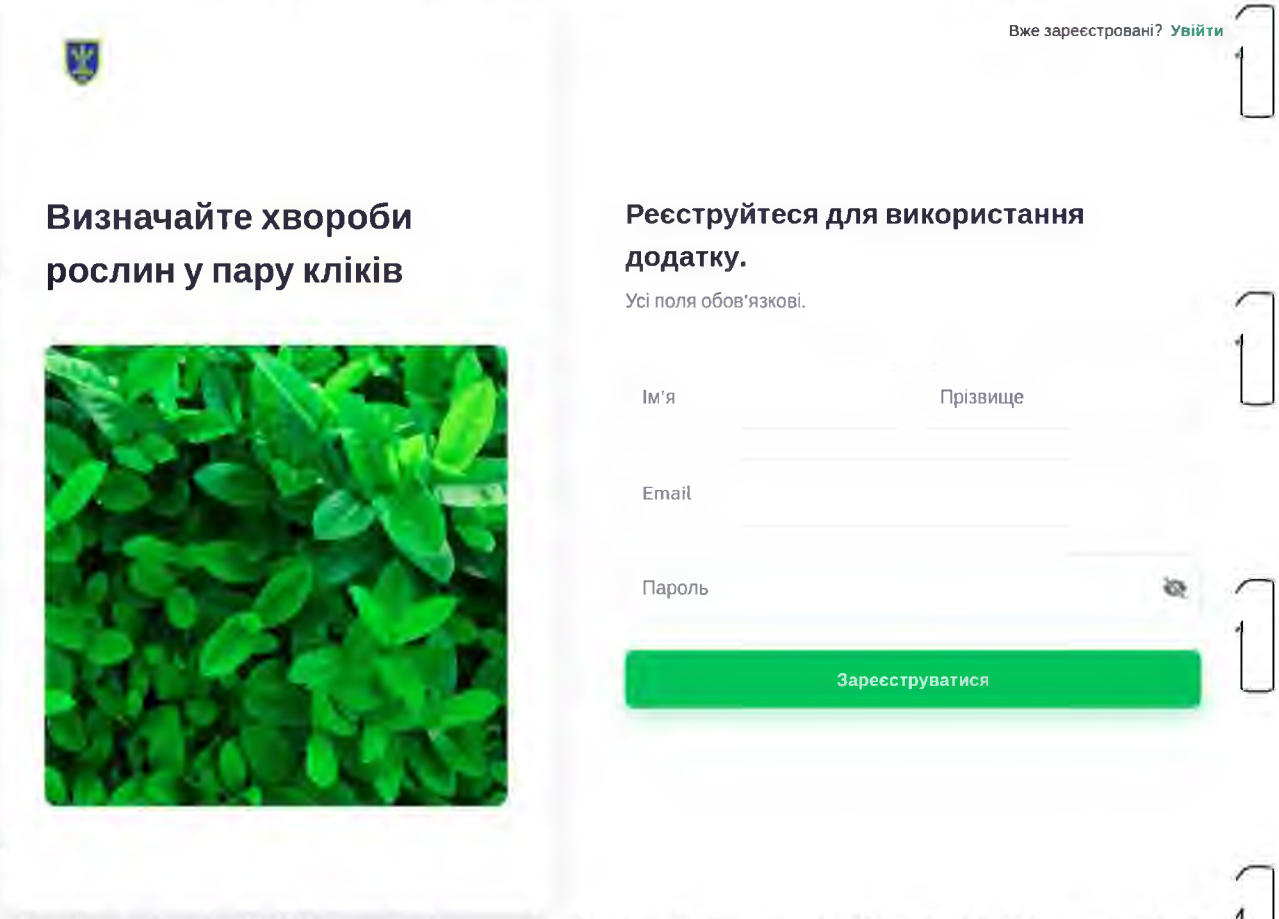


Рисунок 4.1 – Сторінка реєстрації у додатку

У процесі реєстрації усі введені дані валідуються спочатку на фронтенді, з використання бібліотеки Yup, а потім на бекенді. Якщо користувач вкаже недійсную поштову скриньку або занадто простий пароль, він побачить повідомлення про помилку валідації. Також на бекенді перевіряється унікальність поштової адреси.

Після реєстрації користувач може увійти в додаток на сторінці авторизації.

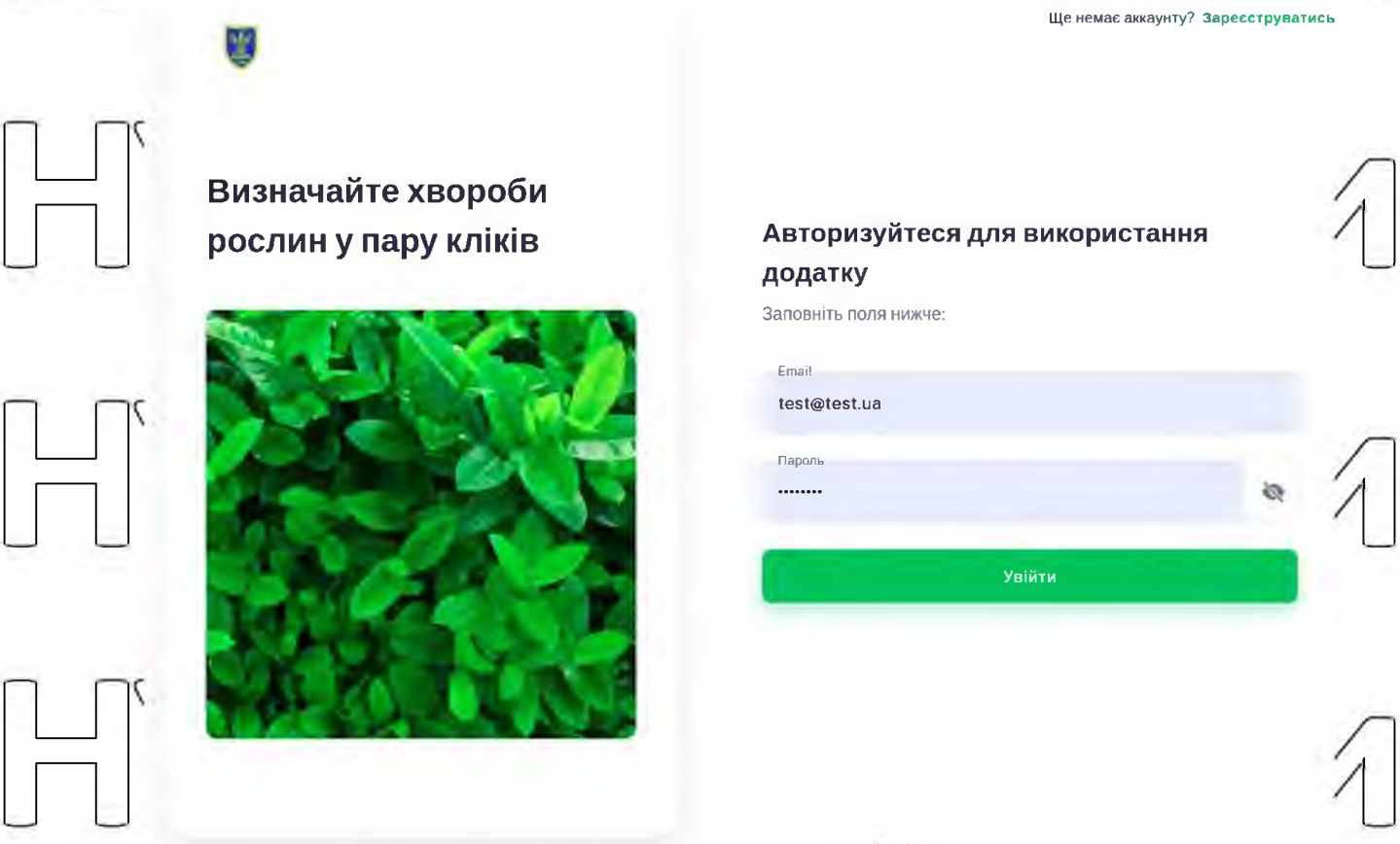


Рисунок 4.2 – Сторінка авторизації

Після авторизації користувачу стають доступні усі функції додатку. Дані користувача зберігаються у LocalStorage для того, щоб при перезавантаженні сторінки не потрібно буде проходити процес авторизації наново.

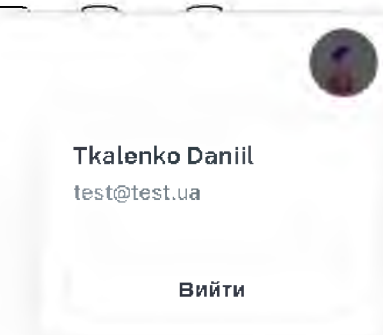
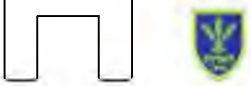


Рисунок 4.3 – Попап з кнопкою виходу



Tkalenko Daniil

Яблуня

Виноград

Картопля

Помідор

Яблуня

Перевірка на можливі захворювання:

- Іржа
- Парша
- Чорна гниль

1 Підготовка зображення

Намагайтеся зробити чітке квадратне зображення листка рослини на одноманітному фоні та в умовах гарного освітлення. Бажано, щоб уся поверхня листа була пряма та певні ознаки хвороби, якщо вони є, були чітко помітні.

Приклад:

Рисунок 4.4 – Інтерфейс користувача

Після авторизації користувач може обрати вид рослини у меню зліва, після чого він зможе використовувати покрокову форму з інструкціями для завантаження та класифікації зображення.

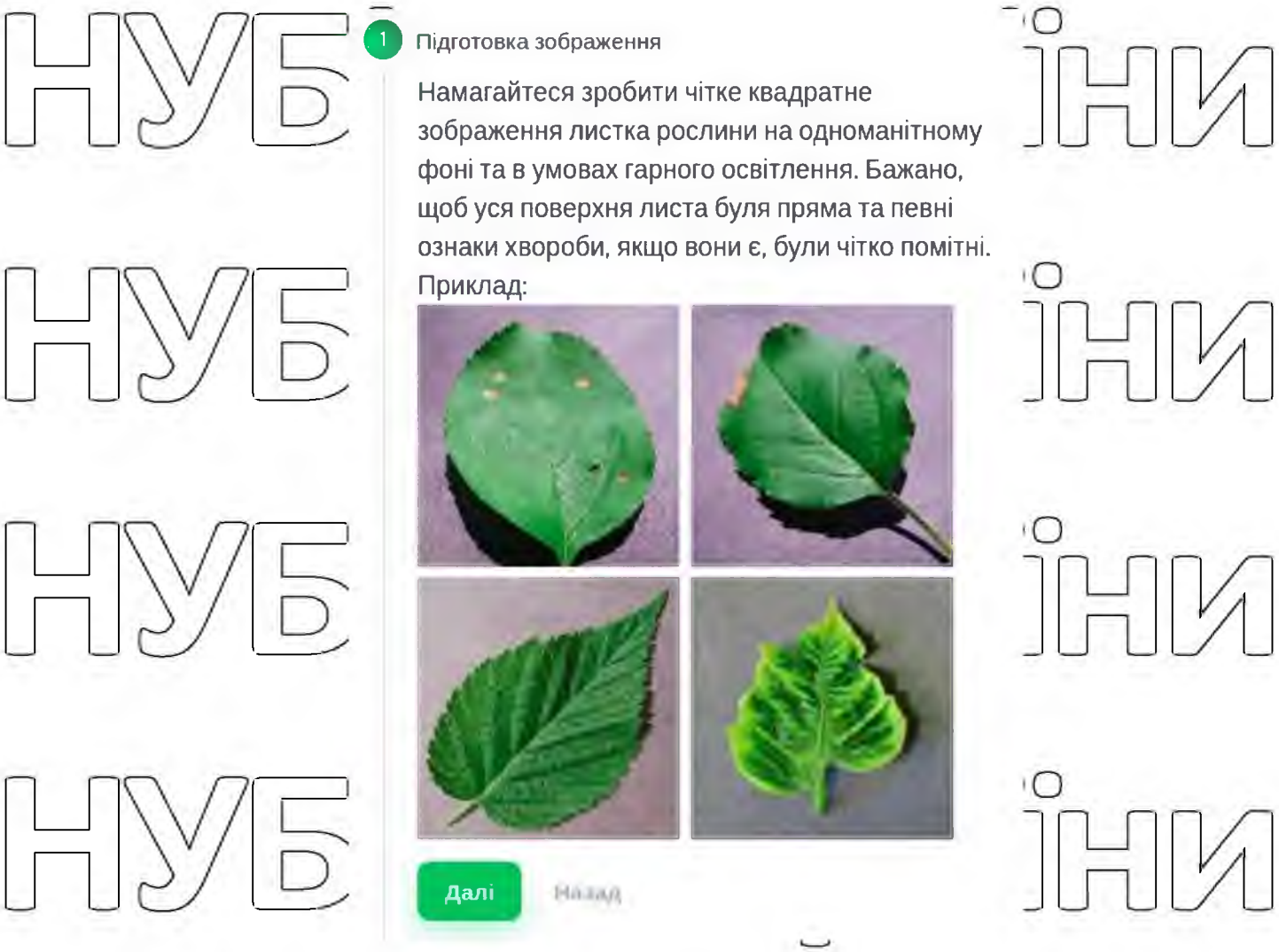


Рисунок 4.5 – Перший крок – підготовка зображення

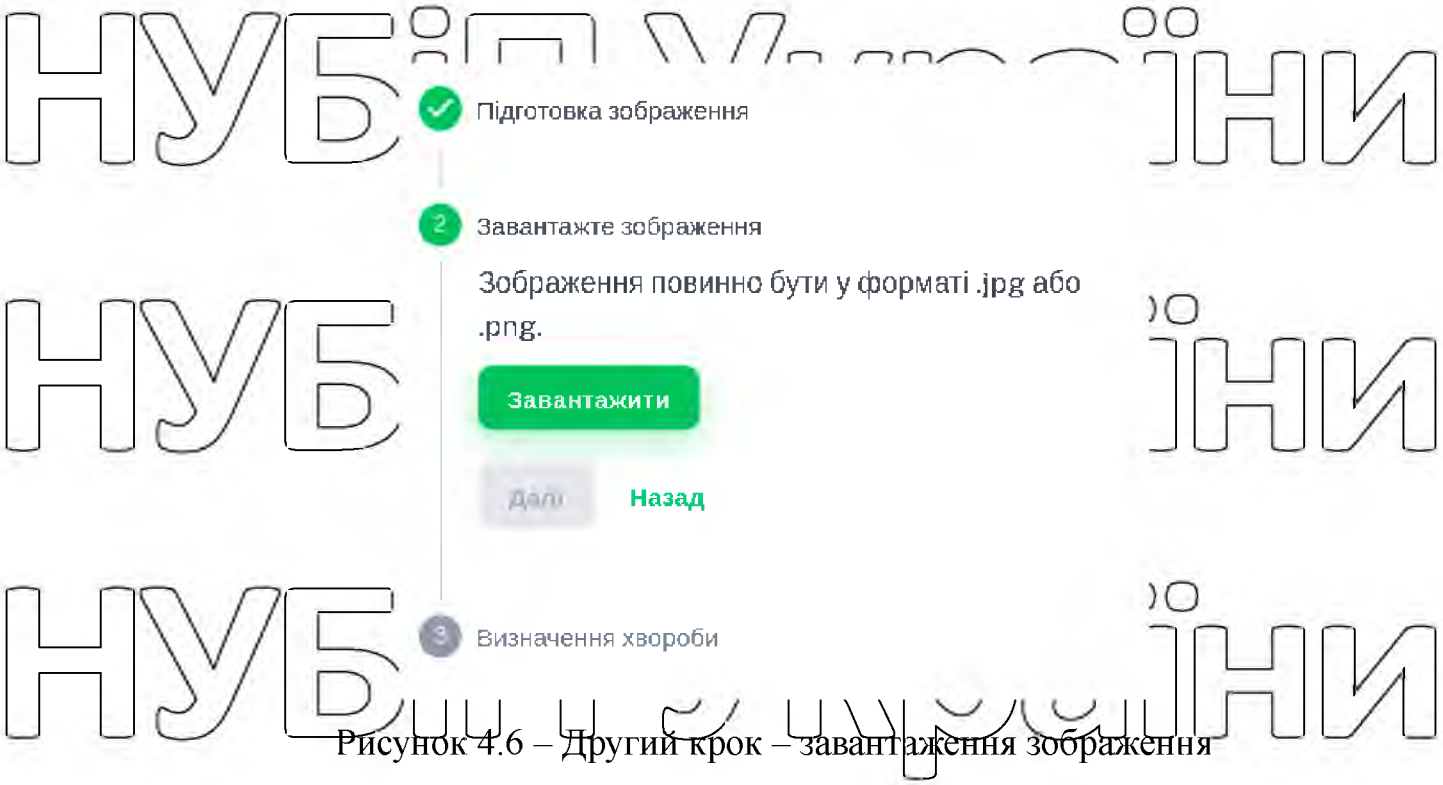


Рисунок 4.6 – Другий крок – завантаження зображення

Поки зображення не було завантажено та валідовано, перейти на наступний етап буде неможливо. Після завантаження фотографії користувач бачить її на екрані та може перейти до наступного кроку.

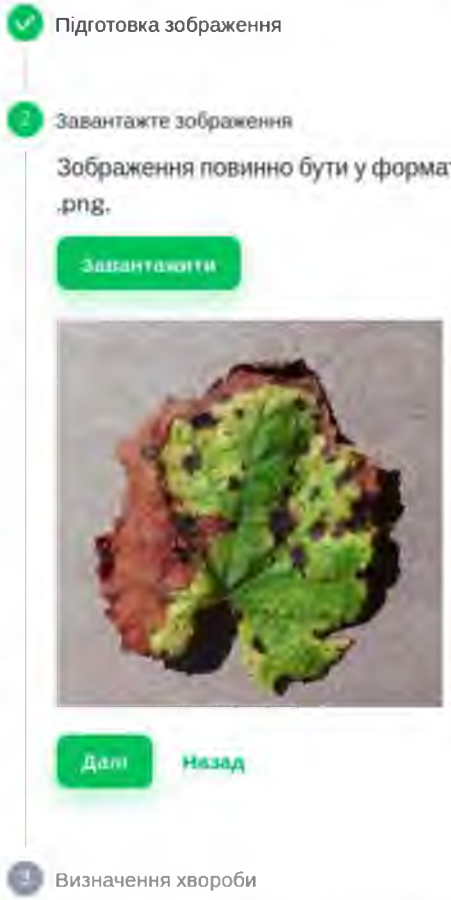


Рисунок 4.7 – Зображення завантажене

Рисунок 4.8 – Результат класифікації зображення

Важливо зазначити, що на скріншотах вище для тестування використовувалися зображення, які не використовувалися для навчання або тестування моделей, отже нейронна мережа працювала з ним вперше, але все одно виконала впевнену та правильну класифікацію.

4.2 Дослідження отриманих моделей

У процесі підготовки до навчання моделі набори даних було розділено на 2 частини — тренувальну та тестову. Навчальні зразки Складають 85% від усіх зразків. Вони використовуються для того, щоб навчити модель розподіляти зразки за класами. Тестові зразки складають 15% всіх зразків. Вони ніколи не використовуються для навчання моделі, проте потрібні під час тестування. Тестові зразки дозволяють перевірити, наскільки добре модель справляється із даними, які обробляє вперше. Саме ця валідаційна вибірка є основою перевірки якості навчання мережі.

Для того щоб вирішити, добре або погано виконує свою роботу навчена мережа, нам необхідна чисельна метрика її якості. У найпростішому випадку такий метрикою може бути частка зразків за якими класифікатор прийняв правильне рішення.

де, P — це загальне число зразків які класифікатор розпізнав вірно, а N — це розмір тренувального набору.

При тренуванні підготовлених мереж для класифікації зображень було вирішено створити 4 окремих мережі для кожного типа рослин. З одного боку це дозволить у майбутньому легко розширювати та масштабувати функціонал додатку, створюючи нові моделі для різних видів рослин, а з іншого — це дає

нам змогу поекспериментувати з параметрами навчання моделей на обрати з них ті, що дадуть кращі результати.

Почнемо з моделі для розпізнавання хвороб картоплі. Для тренування цієї мережі використовувалися наступні параметри:

– кількість епох – 20;
– розмір серії – 128;
– коефіцієнт швидкості навчання – 0,001.

Епоха означає, що модель вивчила кожен приклад із навчальної вибірки хоча б 1 раз. Наприклад, якщо зазначено 20 епох, то під час навчання набір даних обробиться 20 разів.

Нейронні мережі глибокого навчання навчаються за допомогою алгоритму стохастичного градієнтного спуску.

Стохастичний градієнтний спуск — це алгоритм оптимізації, який оцінює градієнт помилки для поточного стану моделі за допомогою прикладів із навчального набору даних, а потім оновлює ваги моделі за допомогою алгоритму зворотного поширення помилок , який також називається просто зворотним поширенням помилки.

Розмір оновлення на який ваги оновлюються під час тренування, називається розміром кроку або швидкістю навчання. Це настроюваний гіперпараметр, який використовується при навчанні нейронних мереж, який має невелике позитивне значення, часто в діапазоні від 0,0 до 1,0.

Швидкість навчання контролює, наскільки швидко модель адаптується до проблеми. Менші темпи навчання вимагають більше епох навчання, враховуючи менші зміни вагових коефіцієнтів кожного оновлення, тоді як більші темпи навчання призводять до швидких змін і потребують меншої кількості епох навчання.

Занадто велика швидкість навчання може призвести до того, що модель занадто швидко сходиться до неоптимального рішення, тоді як надто мала швидкість навчання може призвести до зупинки процесу.

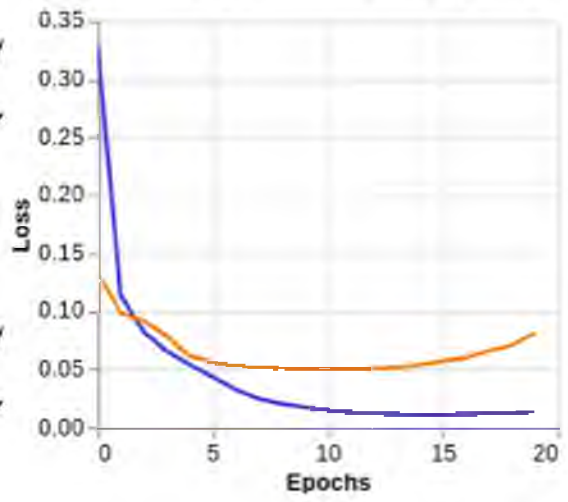
Завдання навчання нейронних мереж глибокого навчання передбачає

ретельний вибір швидкості навчання. Це може бути найважливішим гіперпараметром для моделі.

Розмір сесії або пакету - це розмір набору зразків, яка буде оброблена у однієї ітерації під час навчання. Припустимо, що ми вибрали пакет розміром 16, а у усього у нас 96 зображень. Це означає, що всі дані будуть поділені на шість пакетів: 96 / 16 = 6. Як тільки модель ознайомиться з шістьома пакетами, завершиться одна епоха.

orFlow також надає можливість використовувати інструмент TensorBoard та tfjs-vis для візуалізації процесу тренування та результатів тестування.

Наступний малюнок ілюструє динаміку зміни точності та втрат протягом навчального процесу, а також показує точність кожної категорії при тестуванні на спеціальному верифікаційному наборі. Синя лінія – це дані з навчального набору даних; зазвичай модель швидко навчиться чітко ідентифікувати дані з навчального набору. Мета навчання – дозволити моделі також класифікувати приклади (нові зображення), яких немає у навчальному наборі. Зелені лінії – це нові зображення із набору перевірки; для моделі це повністю нові зображення, але вони зроблені у ідентичних з тренувальними зображеннями умовах.



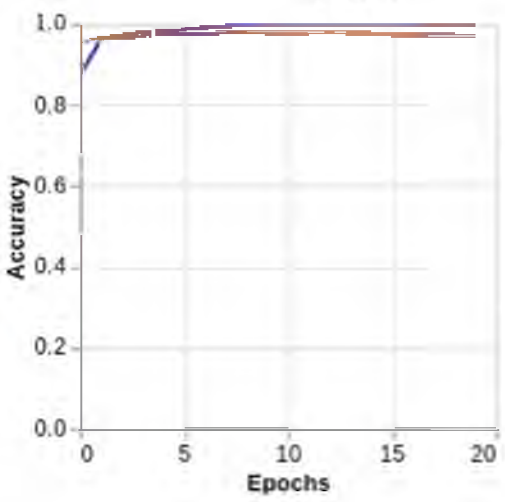
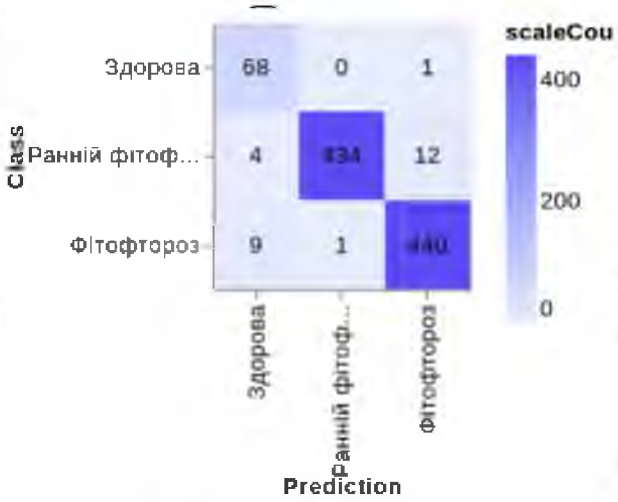


Рисунок 4.9 – Динаміка зміни Accuracy та втрат (картопля)



CLASS	ACCURACY	# SAMPLES
Здорова	0.99	69
Ранній фітофтороз	0.96	450
Фітофтороз	0.98	450

Рисунок 4.10 – Матриця помилок та точність по класам з тестової вибірки (кароиля)

У матриці помилок подано зведені дані про те, наскільки точні прогнози моделі. За цією матрицею можна визначити, у яких класах виникають помилки.

Щоб оцінити, як навчена модель буде працювати з типовими даними, але, яких вона не бачила у процесі тренування, існує інший параметр - accuracy per class. Як бачимо, модель майже безпомилково може класифікувати зображення вказаних хвороб із тестової вибірки.

Далі розглянемо результати моделі по класифікації хвороб томатів. Параметри тренування були наступними:

- кількість епох – 50;
- розмір серії – 32;
- коефіцієнт швидкості навчання – 0.003

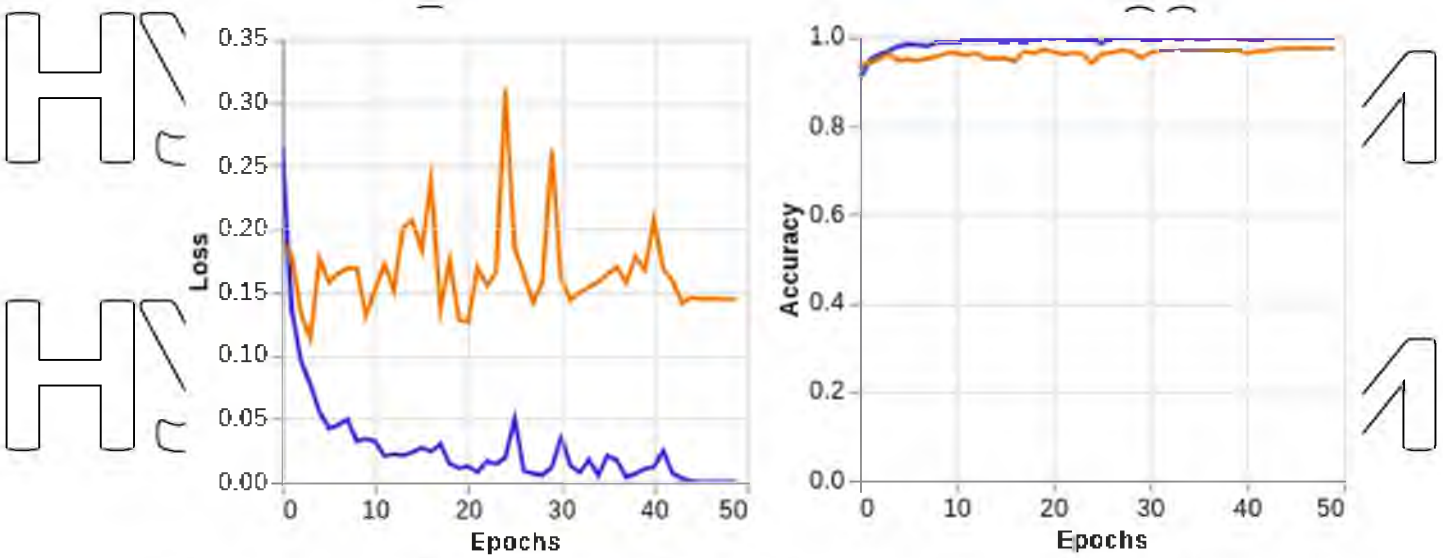


Рисунок 4.11 – Динаміка зміни Accuracy та втрат (помідор)

Рисунок 4.12 – Матриця помилок та точність по класам з тестової вибірки (помідор)

Далі розглянемо результати моделі по класифікації хвороб винограду із застосуванням наступних параметрів навчання:

- кількість епох – 80;
- розмір серії – 128;
- коефіцієнт швидкості навчання – 0,0008.

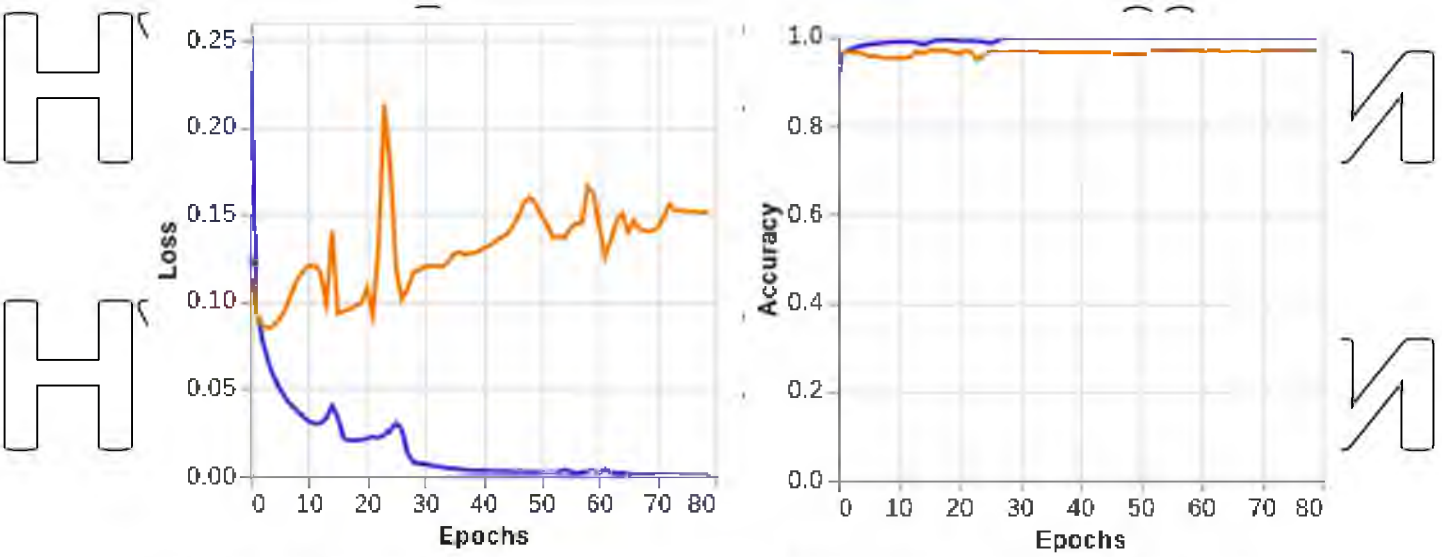
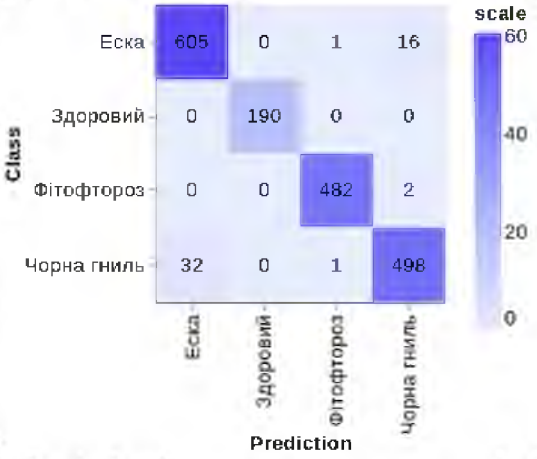


Рисунок 4.13 – Динаміка зміни Accuracy та втрат (виноград)



CLASS	ACCURACY	# SAMPLES
Еска	0.97	622
Здоровий	1.00	190
Фітофтороз	1.00	484
Чорна гниль	0.94	531

Рисунок 4.14 – Матриця помилок та точність по класам з тестової вибірки (виноград)

Останньою буде модель для класифікації хвороб яблунь. Для цієї моделі параметри тренування були наступними:

- кількість епох – 50;
- розмір серії – 64;
- коефіцієнт швидкості навчання – 0,0009.

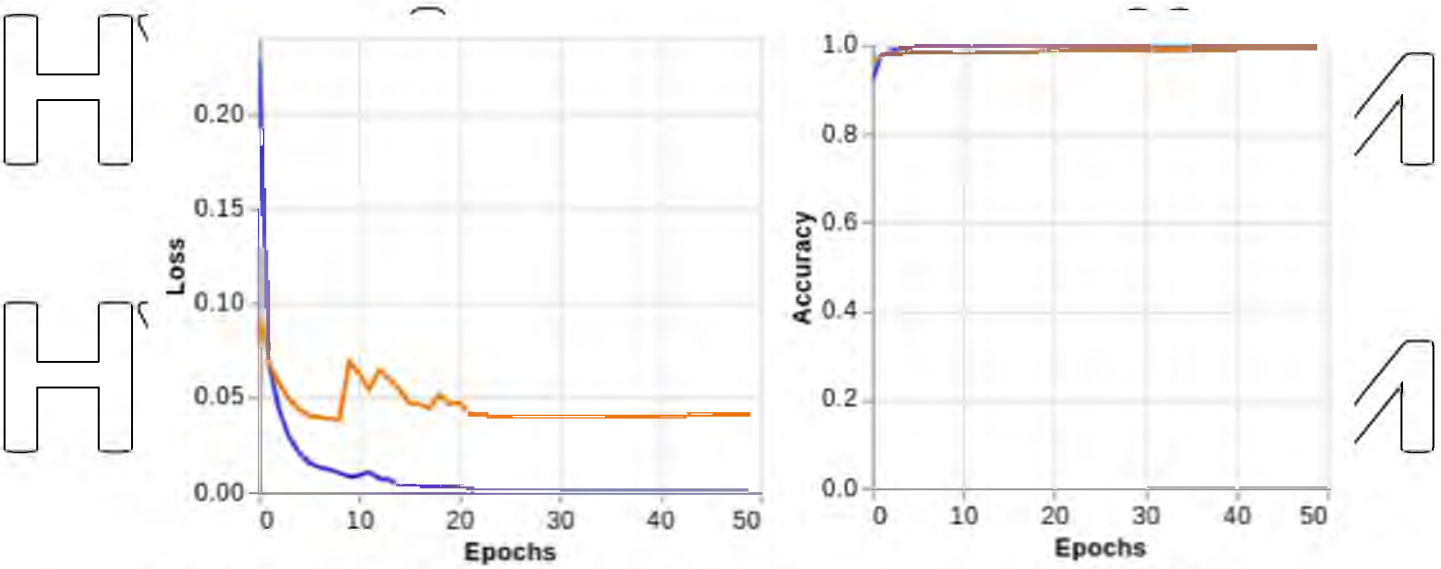


Рисунок 4.15 – Динаміка зміни Accuracy та втрат (яблуня)

Рисунок 4.16 – Матриця помилок та точність по класам з тестової вибірки (яблуня)

У результаті цих експериментів було встановлено, що оптимальними параметрами для представленої моделі є швидкість навчання 0,0009 та кількість епох 50. При цих параметрах навчена модель видає найкращі показники у тестах.

Під час ручного тестування моделі з використанням даних користувача результати можуть сильно відрізнятися від отриманих вище через те, що у наборі даних для тренування та тестування були використані зображення

зроблені у відмінних умовах. Якщо ж використовувати модель з зображеннями листків у інших умовах, результати класифікації будуть значно гіршими. Щоб вирішити цю проблему, слід використовувати більший набір даних з різноманітними зображеннями для навчання. Звичайно, це потребує більших обчислювальних потужностей, але ще більшою проблемою є формування відповідного набору даних.

4.3 Результати дослідження

У цій роботі реалізовані моделі глибокої згорткової нейронної мережі для виявлення та діагностики захворювань у рослин за їх листям, оскільки ЗНМ досягли вражаючих результатів у сфері машинного зору. Реалізовані моделі були навчені з відкритим набором даних, що складається з 4 різних видів рослин і 15 різних категорійних класів хвороб і здорових листків рослин. Щоб оцінити продуктивність моделей, були включені різні параметри, такі як розмір партії, випадання та різна кількість епох. Реалізовані моделі досягли високих показників точності класифікації захворювань, які були значно вищими, ніж у традиційних підходів ручної роботи. У порівнянні з іншими моделями глибокого навчання, впроваджена модель досягла кращої продуктивності з точки зору точності та потребувала менше часу на навчання завдяки технології трансферного навчання. Крім того, архітектура MobileNetV2 сумісна з мобільними пристроями, і беручи до уваги те, що модель буде використовуватися у вигляді веб-додатку це є величезним плюсом. Точність результатів ідентифікації хвороб показала, що глибока модель ЗНМ є перспективною і може значно вплинути на ефективну ідентифікацію хвороб, а також може мати потенціал у виявленні хвороб у сільськогосподарських системах.

Однак на поточному етапі існує ряд обмежень, які необхідно усунути в

подальшій роботі. По-перше, під час тестування на наборі зображень, зроблених в умовах, відмінних від зображень, які використовуються для навчання, точність моделі істотно знижується. Для підвищення точності необхідний більш різноманітний набір навчальних даних. Поточні результати свідчать про те, що більше даних буде достатньо, щоб істотно підвищити точність результатів під час реального використання додатка кінцевими користувачами.

Друге обмеження полягає в тому, що зараз існує обмеження класифікацією одиничних листків, звернених вгору, на однорідному фоні. Хоча це прості умови, які не важко виконати при використанні системи, реальна програма повинна мати можливість класифікувати зображення хвороби, як вона проявляється безпосередньо на рослині, адже багато захворювань проявляються не тільки на верхній стороні листя, а й на багатьох різних частинах рослини. Таким чином, нові зусилля зі збору зображень мають стати головно метою у шляху к створенню повнощінної системи класифікації хвороб рослин.

ВИСНОВКИ

У процесі дипломної роботи було спроектовано, розроблено та досліджено систему, яка використовує нейронну мережу та сучасний веб-інтерфейс для класифікації зображень хвороб рослин.

Під час роботи виконано наступні завдання:

Проаналізовано зростаючу проблему швидкої та надійної діагностики захворювань рослин. Описано варіанти використання машинного навчання та нейронних мереж для вирішення цих завдань, наведено приклади таких рішень та принципи роботи мереж для класифікації зображень.

Спроектовано архітектуру веб-додатку як інтерфейсу для нейронної мережі, обгрунтовані та обрані сучасні засоби машинного навчання у веб-середовищі та засоби для створення веб-інтерфейсів.

Наведено принципи навчання нейронних мереж, описано кроки створення веб-додатку, розглянуто архітектурні рішення під час створення програми та продемонстровано приклади коду для різних частин системи.

4. Проведено дослідження системи, у результаті чого були встановлені оптимальні параметри мережі, а саме кількість епох – 50, розмір пакету – 64 та швидкість навчання – 0,0009. У результаті отримані високі результати точності розпізнавання зображень отриманими моделями – точність класифікації 98%, низький показник втрат – 0,04. Також перевірена працездатність та зручність веб-інтерфейсу програми.

Таким чином, усі поставлені задачі було успішно вирішено.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. P. Ghamisi, J. Plaza, Y. Chen, J. Li, A.J. Plaza Advanced spectral classifiers for hyperspectral images: a review IEEE Geosci Remote Sens Mag, 5, стр.
2. Роббінс, Дженніфер HTML5, CSS3 та JavaScript. Вичерпний посібник (+ DVD-ROM) / Дженніфер Роббінс. – М.: Ексмо. – стр.528, 2014
3. Создание блок-схемы алгоритма при помощи MS Visio URL: https://delphi.ucoz.org/publ/sozdanie_blok_skhemy_algoritma_pri_pomoshhi_ms_visio/1-1-0-91 (дата звернення: 20.05.2021).
4. Справочник HTML URL: https://www.w3schools.com/ (дата звернення:
5. JR Uijlings, KE van de Sande, T. Gevers, AW Smeulders. Selective search for object recognition. International journal of computer vision, 104 (2). стр. 154-171, 2013.

Architecture: Definition, Models, Types, and More URL: https://hackr.io/blog/web-application-architecture-definition-models-types-and-more (дата звернення: 25.04.2021).

op-javascript-machine-learning-libraries-in-2019-cb63b95bdd10 (дата звернення: 14.04.2021).

-tensorflow/ (дата звернення: 09.10.2021).

9. Best Practices for Preparing and Augmenting Image Data for CNNs URL: https://machinelearningmastery.com/best-practices-for-preparing-and-augmenting-image-data-for-convolutional-neural-networks/ (дата звернення:

Сучасний підручник JS URL: https://learn.javascript.ru/ (дата звернення:

es Rendus l'Académie Bulg Des Sci, 68, pp. 175-182, 2015.

R.N. Strange, P.R. Scott Plant disease: a threat to global food security Annu Rev Phytopathol, стр. 83-116, 2005

uction in hyperspectral images J Indian Soc Remote Sens, стр. 157-167, 2018.

14. Разработка требований к программному обеспечению, Русская Редакция, БХВ-Петербург, Карл Вигерс, стр. 213, 2016.

-Hill Education; 8 edition, стр. 263, 2014.