НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

На правах рукопису
УДК 004.93'14

До захисту допущено
Завідувач кафедри ММСА
О.Л.Тимощук

«___»_____2021 р.

Магістерська дисертація

на здобуття ступеня магістра
за спеціальністю 122 Комп'ютерні науки
на тему: «Розпізнавання хвороб сільськогосподарських культур за допомогою комбінації архітектур глибинних нейронних мереж»

Виконала:
студентка ІІ курсу, групи КА-03 мп
Житанська Дар'я Сергіївна

_____

Керівник:
доцент кафедри ММСА,
к.ф-м.н, доц. Шубенкова І.А.

_____

Рецензент:
професор кафедри інформаційних систем та технологій
КПІ ім. Ігоря Сікорського
д.т.н., проф. Корнієнко Б.Я. _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань

Студент _____

Київ
2021

Національний технічний університет України
«Київський політехнічний інститут імені
Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – другий (магістерський)

Спеціальність – 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ О.Л.Тимощук

(підпис)

«___»_____2021 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Житанській Дар'ї Сергіївні

Тема дисертації: «Розпізнавання хвороб сільськогосподарських культур за допомогою комбінації архітектур глибинних нейронних мереж», керівник роботи Шубенкова Ірина Анатоліївна, к.ф.-м.н., доцент, затверджені наказом по університету від «02» листопада 2021 р. № 3651-с.

1. Термін подання студентом дисертації 12.12.2021

2. Об'єкт дослідження: агроринок

3. Предмет дослідження: різноманітні хвороби коренеплідної рослини маніоки, методи машинного навчання.

4. Перелік завдань, які потрібно зробити:

1) Здійснити огляд технічної літератури за темою роботи;

2) Дослідити актуальність обраної теми;

3) Ознайомитись із існуючими методами розпізнавання хвороб сільськогосподарських культур;

4) Здійснити порівняльний аналіз наявних методів, виявити їх переваги та недоліки;

5) Розробити та реалізувати систему, що прогнозувати появу хвороби сільськогосподарських культур через зображення;

6) Провести аналіз результатів;

7) Провести аналіз ринкових можливостей запуску стартап проекту;

8) Розробити концептуальні висновки;

9) Підготувати ілюстративний матеріал;

10) Оформити пояснювальну записку.

5. Орієнтовний перелік ілюстративного матеріалу:

1. Аналіз стану агроринку;
2. Методи, що використовувалися для оцінки стану агроринку;
3. Обґрунтування вибору методу, що був використаний для розв’язання задачі;
4. Приклад результатів використання системи;
5. Розрахункові таблиці з результатами.

6. Дата видачі завдання: 1 вересня 2021 року.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Отримання завдання на магістерську дисертацію	01.09.2021 – 05.09.2021	Виконано
2.	Огляд технічної літератури за темою	06.09.2021 – 12.09.2021	Виконано
3.	Концептуальний вступ дисертації. Формулювання об’єкта, предмета, цілі, завдань, новизни, практичної значущості результатів. Дослідження актуальності обраної теми	13.09.2021 – 19.09.2021	Виконано
4.	Перший розділ. Методи виявлення хвороб листя маніоки на репрезентативних зображеннях	20.09.2021 – 26.09.2021	Виконано
5.	Другий розділ. Огляд моделей нейронних мереж	27.09.2021 – 03.10.2021	Виконано
6.	Третій розділ. Розробка системи підтримки, опис дослідження та аналіз результатів	04.10.2021 – 10.10.2021	Виконано
7.	Розширення функціоналу системи	11.10.2021 – 17.10.2021	Виконано
8.	Аналіз результатів	18.10.2021 – 24.10.2021	Виконано
9.	Проведення аналізу ринкових можливостей стартап – проекту	25.10.2021 – 31.10.2021	Виконано
10.	Підготовка ілюстративного матеріалу	01.11.2021 – 14.11.2021	Виконано
11.	Оформлення пояснювальної записки	15.11.2021 – 26.11.2021	Виконано

Студент _____ Д.С. Житанська

Науковий керівник дисертації _____ І.А. Шубенкова

РЕФЕРАТ

Магістерська дисертація: 112 с., 41 рис., 23 табл., 19 джерел.

Об’єктом дослідження є агроринок.

Предметом дослідження є різноманітні хвороби коренеплідної рослини маніоки, методи машинного навчання.

Мета дослідження – дослідити існуючі реалізації технологій розпізнавання хвороб за допомогою зображення та розробити систему підтримки, яка буде базуватись на методах машинного навчання.

Результат роботи: розроблено та реалізовано систему, що дозволяє оцінити майбутній стан урожаю за поточним виглядом рослини.

Новизна роботи: розроблена система надає оцінку стану коренеплідної рослини і вказує на можливі розвитки хвороби.

АГРОРИНОК, МЕТОД *K*-НАЙБЛИЖЧИХ СУСІДІВ, МЕТОД ОПОРНИХ ВЕКТОРІВ, ГЛИБОКЕ НАВЧАННЯ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, НЕЙРОННІ МЕРЕЖІ.

ABSTRACT

Master’s Thesis 112 pp., 41 Fig., 23 tables, 19 sources.

The object of research is the agricultural market.

The subject of research is various diseases of the cassava root plant, machine learning methods.

The purpose of the study is to investigate the existing implementations of disease detection technologies using images and to develop a support system that will be based on machine learning methods.

The result of work: developed and implemented a system that allows you to assess the future state of the crop according to the current type of plant.

The novelty of the work: the developed system provides an assessment of the condition of the root plant and indicates the possible development of the disease.

AGRICULTURAL MARKET, *K*-NEAREST NEIGHBORS, SUPPORT VECTOR MACHINE, DEEP LEARNING, CONVOLVED NEURAL NETWORKS, NEURAL NETWORKS.

ЗМІСТ

ВСТУП

Маніок (касава) - другий за величиною виробник вуглеводів африканського континенту. Це одна з найважливіших основних сільськогосподарських культур Африки, збагачена білком і великим вмістом вітамінів, використовується як основний замінник рису. Згідно з даними Центрального статистичного агентства міста Бандар-Лампунг за 2015-2017 роки, виробництво маніоки посідає друге місце із загальним виробництвом 5323,00 тонн після рисових заводів із загальним виробництвом 29 583,68 тонн. Проте виробництво маніоки в 2018 році зменшилося на 34,5% порівняно з попереднім роком, що пов'язано зі зменшенням площі врожаю. І хоча ця рослина відома своєю здатністю протистояти суворим умовам навколишнього середовища, поширення спалахів хвороб часто загрожує врожаю та становлять серйозну загрозу для натуральних фермерів, які їх вирощують. У той час як понад 80% невеликих домашніх господарств у країнах Африки на південь від Сахари вирощують цей корінь, лише небагато з них здатні виявляти й пом’якшувати руйнівні наслідки спалахів захворювань, якими вони регулярно страждають. Наразі, щоб оцінити, чи уражені рослини хворобами, фермери повинні співпрацювати з представниками місцевого самоврядування, щоб залучити сільськогосподарських експертів для особистого огляду рослин або проводити певні лабораторні дослідження. Лабораторне дослідження або допомога фахівця з рослин зазвичай проводиться для виявлення хвороби в листках маніоки. Листя, уражені хворобою, впливають на врожайність сільськогосподарських культур, оскільки листя є важливою частиною рослини, яка функціонує як місце для фотосинтезу. Тканина флоеми транспортує результати фотосинтезу до всіх інших частин рослини. Якщо листя рослини здорові і процес фотосинтезу здійснюється правильно, то ріст стебел і бульб також ідеальний. Але якщо

листя уражені хворобами, порушується процес фотосинтезу, порушується також ріст стебел і бульб, що спричиняє низьку якість врожаю. На жаль, такий процес оцінки є надзвичайно трудомістким, повільним та неефективним, що ставить фермерів під більший ризик втрати більшої частини свого врожаю, якщо вони справді мають справу зі спалахом хвороб серед своїх рослин. Щоб пришвидшити цей процес і надати фермерам найкращу можливість зберегти свої врожаї, маємо на меті допомогти розробити таку згорткову (конволюційну) нейронну мережу, яка зможе швидко й точно класифікувати, чи страждає дана рослина хворобою чи ні, просто надавши моделі зображенням листя цієї рослини. З огляду на те, що багато камер, до яких мають доступ місцеві фермери, можуть бути досить низької якості, ми будемо використовувати репрезентативні фотографії для навчання нашої моделі та спробуємо створити рішення, яке буде надійним у своїй ефективності та здатним працювати для всіх тих, хто так відчайдушно потрібно це.

Набір даних, який будемо використовувати для створення та оптимізації нашої мережі, містить 21 397 спостережень за зображеннями листків маніоки та їх відповідні класифікації хвороб. П’ять категорій, до яких підпадають зображення листя: бактеріальна хвороба маніоки (CBB), хвороба бурої смуги маніоки (CBSD), маніока зелена плямистість (CGM), хвороба мозаїки маніоки (CMD) і здорова.

Метод, що запропонований у цьому дослідженні як вирішення задачі, є згорткова нейронна мережа (CNN). Він має найбільш значущі результати у розпізнаванні зображень. Це тому, що метод CNN має можливість обробляти двовимірні дані/зображення. Мережі на CNN мають спеціальний шар, який називається згортковим шаром. У шарі згортки введення у вигляді зображення створює візерунок з кількох частин зображення, щоб його було легше класифікувати. Виходячи з цього, метод CNN може зробити функцію навчання зображення більш ефективною для реалізації.

1 МЕТОДИ ВИЯВЛЕННЯ ХВОРОБ ЛИСТЯ МАНІОКИ НА РЕПРЕЗЕНТАТИВНИХ ЗОБРАЖЕННЯХ

1.1 Поняття та різновид хвороби листя маніоки

Маніока (Manihot esculenta) — це багаторічний чагарник родини молочайних, який вирощується в основному для зберігання коренів, які вживають в їжу як овоч. Їстівні коріння рослини, як правило, циліндричні і конічні, мають білий, бурий або червонуватий колір (рис. 1.1).

Рисунок 1.1 – Корінь маніоки

Маніока — деревна рослина з прямостоячими стеблами і спірально розташованими простими лопатевими листками з черешками (стеблами) до 30 см завдовжки (рис. 1.2). Рослина утворює на китиці квітки без пелюсток. Рослини маніоки можуть досягати 4 м у висоту, і зазвичай їх збирають через 9-12 місяців після посадки.

Рисунок 1.2 – Листя маніоки

Маніоку також називають бразильською марантою, юкою або тапіокою, а походження рослини невідоме. Невідомо чи рослина зустрічається в дикому вигляді, але, можливо, вперше була культивована в Бразилії. Маніок — третє за величиною джерело харчових вуглеводів у тропіках, після рису та кукурудзи. Це основний продукт харчування в країнах, що розвиваються, забезпечуючи основним харчуванням понад півмільярда людей. Це одна з найбільш посухостійких культур, здатна рости на маргінальних ґрунтах.

Маніока процвітає в тропічних і субтропічних регіонах світу, оскільки для оптимального зростання їй потрібна тепла температура (рис. 1.3). Рослинам потрібна принаймні 8 місяців теплої погоди, вони процвітають у регіонах з теплим, вологим кліматом з регулярними опадами. Маніоку можна вирощувати на багатьох типах ґрунту, навіть у бідному ґрунті. Вона стійка до посухи, але не переносить перезволоження. Утворення коренів збільшується при температурі від 25 до 32°C. Живці маніок розмножують зі стеблових живців, оскільки бульби не дають бутонів. Стеблові живці слід брати тільки з рослин, які не хворіють, мають вік не менше 10 місяців і мають бульби [1].

Рисунок 1.3 – Плантації маніоки

Розглянемо які ж хвороби можуть зашкодити цій рослині та як їх можна виявити за допомогою поверхневого дослідження. Спочатку подивимось на повністю здорове листя маніоки (рис. 1.4).

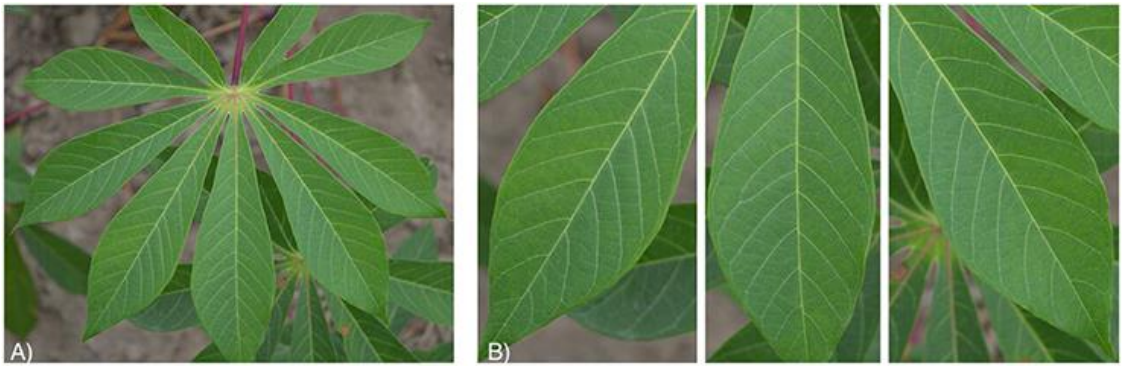


Рисунок 1.4 – Здорове листя маніоки

П'ять симптомів захворювання, що досліджуємо та появи шкідників такі:

1. CBSD (cassava brown streak disease) є результатом зараження іпомовірусами бурої смужки маніоки (CBSI) (родина Potyviridae, рід Ipomovirus). Існують два види, пов’язані з хворобою, вірус маніоки

коричневої смуги (CBSV) та вірус маніоки коричневої смуги (UCBSV). Ці два види вірусів переносяться білокрилками [Bemisia tabaci (Genn.)] напівпостійним способом. При зараженні листя маніоки демонструють плямисту жовтизну, яка зазвичай починається з вторинних вен і прогресує до третинних вен у міру посилення інфекції (рис. 1.5). Цей жовтуватий хлороз поширюється по жилах, поки сильно уражені листя не стають переважно жовтими. Симптоми захворювання можуть відрізнятися залежно від сорту, віку рослини та погодних умов. Сорти та рослини в молодому віці можуть бути заражені, але безсимптомно. Ці два віруси також можуть викликати коричневі смуги на стеблах заражених рослин і буре некротичне гниття в коренях бульб, що може зробити їх неїстівними.

Рисунок 1.5 – Хвороба CBSD листя маніоки

2. CMD (cassava mosaic disease) є результатом зараження бегомовірусами мозаїки маніоки (CMBs) (сімейство Geminiviridae, рід Begomovirus). Існує багато видів і рекомбінантних штамів, пов’язаних з цією групою вірусів, хоча поширеною формою в прибережній Східній Африці, де відбирали проби, є вірус мозаїки східноафриканської маніоки (EACMV) (Ndunguru et al., 2005). Види вірусу переносяться B.

tabaci (Genn.) постійно, на відміну від CBSI. Повторно заражені рослини починають проявляти симптоми зверху, тоді як рослини, заражені через посаджений живець, часто проявляють симптоми у всіх листках. Симптоми CMD є типовою мозаїкою, в якій є суміш жовтих/блідо-зелених хлоротичних плям і зелених зон (рис. 1.6). На відміну від CBSD, листя зазвичай спотворені за формою, а там, де симптоми серйозні, розмір листя значно зменшується, а рослина затримується. Затримка росту та пошкодження хлорофілу внаслідок хлорозу призводить до кількісного зниження врожаю.

Рисунок 1.6 – Хвороба CMD листя маніоки

3. BLS (brown leaf spot) викликається грибом [Mycosphaerella henningsii (Sivan)]. Цей гриб поширений по всьому світу і зазвичай не спричиняє значних втрат врожаю. Хвороба проявляється коричневими круглими плямами на листі, а деякі різновиди мають хлоротичний ореол навколо плям (рис. 1.7) . Сильні інфекції можуть призвести до того, що листя пожовтіє або коричневіє. Кругові плями можуть висихати і тріскатися залежно від навколишнього середовища.

Рисунок 1.7 – Хвороба BLS листя маніоки

4. GMD (cassava green mite damage) викликається зеленими кліщами маніоки [Mononychellus tanajoa (Bondar)]. Це широко поширений шкідник в Африці та Південній Америці. Кліщі викликають невеликі білі, схожі на подряпини плями на місцях, де вони харчувалися, а у важких випадках викликають покриття всього листа візерунком (рис. 1.8). Існує таке зменшення вмісту хлорофілу, що лист може затриматись у спосіб, подібний до того, що викликається CMD. Залежно від сорту та середовища, зараження може призвести до втрати врожаю бульбових коренів до 30%.

Рисунок 1.8 – Хвороба GMD листя маніоки

5. RMD (Red mite damage) викликається червоним павутинним кліщем маніоки [Oligonychus biharensis (Hirst)], який широко поширений по всій Африці. Їх пошкодження при харчуванні також спричиняють невеликі плями, схожі на подряпини, на листі, але зазвичай дають виразний червонувато-коричневий колір іржі. Підживлення також зосереджено навколо основної жилки, але серйозні інвазії можуть призвести до того, що весь лист стане помаранчевим (рис. 1.9).

Рисунок 1.9 – Хвороба RMD листя маніоки

Хоча GMD і RMD не є суто хворобами, для простоти ми посилаємось на всі умови, що впливають на рослини, які розглядалися в цьому дослідженні як хвороби.

Як видно з зображень вище, ознаки, що вказують на хворобу є тонкими і здатність розрізняти ці різні симптоматичні прояви та точно діагностувати рослини вимагає певного рівня досвід та навички, якими більшість не володіють. На щастя, саме ці відносно ненав’язливі особливості нейронні мережі добре вміють аналізувати та виявляти зв’язки, які пов’язують їх із пов’язаними класами. Однією з причин таких потужних інструментів глибокого навчання є те, що незалежно від знань та компетентності людини, яка використовує модель, хороша нейронна мережа може вийти за межі будь -якого формального навчання, яке знадобиться людині, щоб розпізнати ці

особливості, як ознаку конкретного класу та розвивати здатність окреслювати межі абстрактних категорій просто на основі свого попереднього досвіду з набором навчальних даних. Будемо використовувати інтелектуальну систему, яка може вирішити ці проблеми, як експерт з інтерактивним дисплеєм, використовуючи програмне забезпечення графічного інтерфейсу користувача (GUI) Python, яке спрямоване на спрощення роботи.

Вже було проведено кілька досліджень, в яких використовується штучний інтелект, наприклад, визначення флавоноїдних сполук з екстракту листя гуави за допомогою адаптивної нейронної системи нечіткого висновку (ANFIS) [3]. Крім того, методи, що використовуються для виявлення хвороби, використовують такі зображення, як аргументація на основі випадку, метод Демпстера Шейфера, спектральні дані, а також сегментація зображень та м’які обчислення. Однак ці методи не здійснюють процес навчання у наборі даних, тому необхідний метод, який може здійснювати процес навчання у наборі даних та змінювати мережу з метою досягнення високих результатів[2].

1.2 Аналіз репрезентативних зображень

Коли потрібно створювати модель, яка дозволить отримати точну та надійну класифікацію, спочатку потрібно ознайомитися з даними, які є у нашому розпорядженні. Крім того, якщо правильно проаналізувати набір даних та виявити важливі ознаки це стане міцним фундаментом дослідження. Навчальний набір складається з 21 397 пар зображень листя маніоки разом із відповідною класифікацією хвороб.

Як згадувалося раніше, якість цих зображень різко відрізнялася і це було головною проблемою. Одна з характеристик, яка була однаковою для всіх зображень, полягала в тому, що всі їхні розміри були 800x600 (ширина на висоту), що зробило зручнішим змінювати розмір усіх зображень із

збереженням співвідношення, не боячись спотворення чи деформації будь-якого із зображень. Також вдалось залишити взаємозв'язки між формами і кольорами на зображенні. З метою економії оперативної пам’яті та зменшення розміру вхідних розмірів моделі, зменшимо висоту та ширину до однієї четвертої від їх початкового розміру (200x150), це дозволить зменшити записи у відповідному представленні масиву на шістнадцяту.

Зображення були перевірені на супутні інфекції, щоб обмежити кількість зображень із кількома захворюваннями. Цей набір даних, який називається «оригінальний набір даних маніоки», містив 2756 зображень. Потім ці фотографії були вручну обрізані в окремі листівки для створення другого набору даних. Цей набір даних, званий «набором даних про листівки маніоки», містив 15 000 зображень листівок маніоки (2 500 зображень на клас). Обидва набори даних були протестовані, щоб знайти кореляцію продуктивності моделі із зображеннями повних листків, але меншою кількістю зображень порівняно з обрізаними листками з більшою кількістю зображень. Основне припущення полягало в тому, що обрізані зображення листя покращать продуктивність моделі, щоб правильно ідентифікувати захворювання, оскільки набір даних був більшим. Обидва набори даних містили шість міток класів, призначених вручну на основі польових діагнозів експертів із захворювання маніоки.

Хоча це не особливо глибокий аналіз, він дає нам деяку цінну інформацію, яка допоможе нам встановити наш мажоритарний класифікатор нижче. З цих даних ми отримали результат, що класи CBSD, CGM та Healthy, мають від 2100 до 2600 зображень для кожного з них, та понад 13000 зображень уражених CMD рослин маніоки, що становить майже 61,5% усього нашого набору дані навчання. Хоча це може бути результатом деякої менш ніж ідеальної вибірки для створення навчального набору, розумно припустити більш вірогідний сценарій, що CMD є, безумовно, найпоширенішою умовою, і це дозволить нам створити дуже наївну базову модель, яка просто класифікує

всі зображення як уражені ним. Цей підхід не є найбільш інформативним, але він забезпечує точність базового рівня, який маємо покращити[3].

1.3 Методи навчання нейронних мереж для класифікації

1.3.1 Метод опорних векторів для задач класифікації

Нехай є навчальна вибірка (X, ***t***) = $\{x_n, t_n\}_{n=1}^{N}$ обсягу N, де $x_n \in R^d$ - ознаковий опис об'єкта, $t_n \in \{-1, +1\}$ – мітка класу, що приймає одне з двох можливих значень. Завдання полягає в тому, щоб на основі навчальної вибірки спрогнозувати мітку класу ***t*** для нового об'єкта, заданого своєю описаною ознакою ***x***.

Метод опорних векторів (Support Vector Machines, SVM) відноситься до сімейства лінійних класифікаторів. У цьому сімействі рішення про належність об'єкта ***x*** до класу ***t*** приймається за знаком лінійного вирішального правила:

$$f(x) = \sum_{j=1}^{d} w_j x(j) + b = w^T x + b,$$

$$t(x) = \begin{cases} +1, & f(x) \geq 0 \\ -1, & f(x) < 0 \end{cases} \qquad (1.1)$$

Тут $w_j \in$ R - деякі ваги, $b \in$ R - параметр зсуву. З геометричної точки зору лінійний класифікатор є певною роздільною гіперплощиною ***f (x)*** = 0 в просторі ознак R^d, при цьому об'єкт належить до першого класу, якщо він лежить з позитивної сторони від гіперплощини, і відноситься до другого класу в іншому випадку.

Вирішальне правило виду (1.1) може бути застосовано тільки для завдання класифікації з двома класами. Розглянемо еквівалентний запис

вирішального правила (1.1), який можна узагальнити на випадок більш складної структури безлічі прогнозів T:

$$t(x) = \arg \max_{t \in \{-1,+1\}} h(x,t) = \arg \max_{t \in \{-1,+1\}} tf(x) \qquad (1.2)$$

Очевидно, що вирішальні правила (1.1) і (1.2) еквівалентні. Вирішальне правило типу (1.2), зокрема, підходить для випадку класифікації на *K* класів. Введемо для кожного класу свою лінійну функцію $f_k(x),\ k = 1, \dots, K$. Тоді вирішальне правило можна записати як

$$t(x) = \arg \max_{t \in \{1,..,K\}} h(x,t) = \arg \max_{t \in \{1,..,K\}} f_t(x) \qquad (1.3)$$

Розглянемо спочатку випадок, коли навчальна вибірка *(X,* ***t)*** є лінійно нероздільні, тобто існують ***w,*** *b* такі, що

$$\begin{aligned} w^T x_n + b > 0, \qquad t_n = 1, \\ w^T x_n + b < 0,\ t_n = -1 \end{aligned} \qquad (1.4)$$

В цьому випадку провести гіперплощини, що коректно розділяє дані, можна різними способами. Визначимо зазор між класом і гіперплощиною, як мінімальна відстань між гіперплощиною і об'єктом класу. Позначимо через d^+і d^- відстань між гіперплощиною і першим і другим класом відповідно (рис. 1.10).

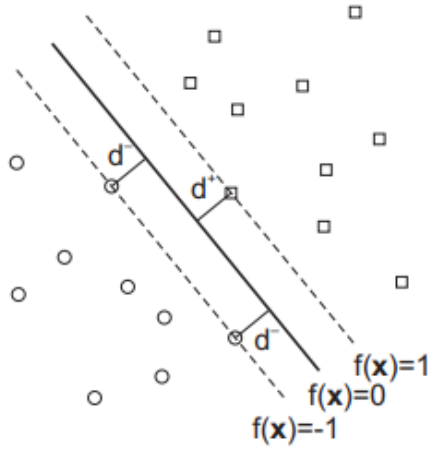


Рисунок 1.10 – Відстань між гіперплощиною та класами

Тоді визначимо оптимальну гіперплощину як гіперплощину, що максимізує відстань:

$$\min(d^+, d^-) \rightarrow \max_{w,b} \tag{1.5}$$

Максимізація відстані між гіперплощиною і даними дозволяє сподіватися на хорошу узагальнюючу здатність в тому випадку, коли тестова вибірка є невеликий варіацією навчальної.

Висловимо величину відстані d^+і d^- через параметри гіперплощини w, b. Розглянемо лінії рівня $f(x) = a$, що проходять через найближчі об'єкти класів до гіперплощини (пунктирні лінії на рис. 1.10). Очевидно, що при фіксованому напрямку гіперплощини оптимальна гіперплощина проходить по середині між цими лініями рівня. Таким чином, можна вважати, що $d^+(w) = d^-(w)$. Зауважимо, що гіперплощина $w^T x + b = 0$ визначена з точністю до масштабу шкали вимірювання w і b. Дійсно, якщо помножити w і b на одне і теж число, то безліч точок x: $w^T x + b = 0$ не зміниться. Однак, при такому множенні лінії рівня $w^T x + b = a$ переміщаються. Вимагатимемо, щоб лінії рівня, що проходять через найближчі об'єкти класів до гіперплощини, визначалися як $w^T x + b = 1$ і $w^T x + b = -1$ (так можна зробити, тому що міркуванням вище

оптимальна гіперплощина завжди проходить по середині між цими двома лініями рівня). Ця вимога однозначно фіксує шкалу вимірювання для w і b.

Розглянемо довільний вектор u_1, що належить гіперплощині, і довільний вектор u_2, що належить лінії рівня $w^T x + b = 1$. Очевидно, що відстань d^+ дорівнює довжині проекції вектора $u_2 - u_1$ на вектор нормалі w (див. рис. 1.11).

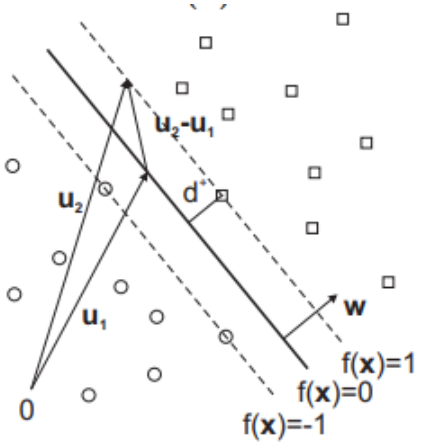


Рисунок 1.11 – Проекція $u_2 - u_1$ на вектор w

Тоді:

$$w^T x + b = 0, w^T x + b = 1 \rightarrow w^T(u_2 - u_1) = 1 \rightarrow \frac{1}{\|w\|} w^T(u_2 - u_1) =$$
$$= pr_w(u_2 - u_1) = \frac{1}{\|w\|} \quad (1.6)$$

Таким чином, відстань $d^+ = d^- = 1/||w||$. Тепер задачу максимізації відстані (1.5) з урахуванням коректного розділення даних можна записати так:

$$\begin{cases} \frac{2}{\|w\|} \rightarrow \max\limits_{w,b}, \\ w^T x_n + b \geq 1, t_n = 1, \\ w^T x_n + b \leq -1, t_n = -1 \end{cases} \approx \begin{cases} \frac{1}{2}\|w\|^2 \rightarrow \min\limits_{w,b}, \\ t_n(w^T x_n + b) \geq 1 \end{cases} \quad (1.7)$$

Видно, що в еквівалентному переході до ||w|| було додано квадрат. Це не змінює рішення задачі, але робить його набагато легше, так як тепер потрібно мінімізувати випуклу функцію.

Розглянемо випадок довільних даних. В такому випадку умова $t_n(w^T x + b) \geq 1$ не може бути виконана для всіх об'єктів. Тоді додаймо умову послаблюючих коефіцієнтів $\gamma_n \geq 0$:

$$t_n(w^T x + b) \geq 1 - \gamma_n \tag{1.8}$$

Очевидно, що можливі три ситуації (рис. 1.12):

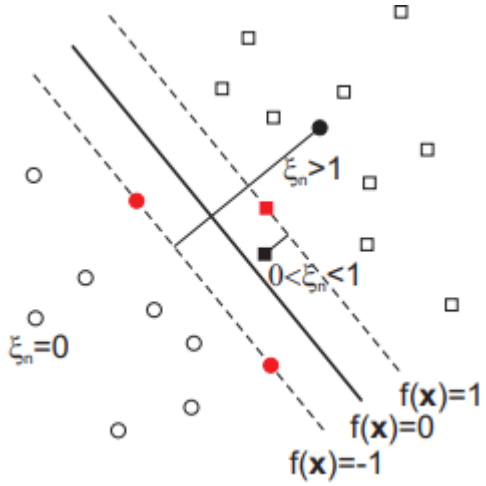


Рисунок 1.12 – Послаблюючі коефіцієнти γ_n

1. $\gamma_n = 0$, помилки немає, x_n лежить за лініями рівня |*f(x)*| = 1.
2. $0 < \gamma_n \leq 1$, помилки немає, x_n лежить у межах $0 \leq t_n f(x) < 1$.
3. $\gamma_n > 1$, помилка є, величина помилки пропорціональна відстані від об'єкта x_n до гіперплощини.

Модифікуємо критерій оптимізації у задачі (1.7), додавши до нього мінімізацію числа помилок у виборці:

$$\frac{1}{2}\|w\|^2 + C\sum_{n=1}^{N}\gamma_n \rightarrow \min_{w,b,\gamma},$$

$$t_n(w^T x + b) \geq 1 - \gamma_n \,, \qquad (1.9)$$
$$\gamma_n \geq 0$$

Тут $C \geq 0$ - коефіцієнт регуляризації, який визначає компроміс між кількістю помилок на навчальній вибірці і простотою лінійного вирішального правила (близькість ваг w_j до нуля).

Таким чином, метод опорних векторів полягає в побудові розділяючої гіперплощини за допомогою рішення задачі оптимізації (1.9). При цьому коефіцієнт регуляризації C задається користувачем до початку навчання. Після навчання прогнозування мітки класу для нового об'єкта $\boldsymbol{x}$ відбувається за схемою (1.1) або еквівалентно (1.2) [7].

1.3.2 Метод k-найближчих сусідів для задач класифікації

Метод заснований на припущенні про те, що близьким об'єктам в просторі ознак відповідають схожі мітки. Для нового об'єкта x метод передбачає знайти найближчі до нього об'єкти $\{x_1, x_2, \ldots, x_k\}$ і побудувати прогноз по їх міткам.

Коефіцієнт K - підбирається по крос-валідації, метрика - вибирається виходячи з обраного простору ознак. Межі класів при цьому будуть дуже складними, що інтуїтивно суперечить тому, що параметр у методу всього один. Парадокс вирішується тим, що насправді об'єкти навчальної вибірки також є своєрідними параметрами алгоритму.

Концепцію пошуку найближчих сусідів можна визначити як «процес пошуку найближчої точки до точки входу з заданого набору даних». Алгоритм зберігає всі доступні випадки (тестові дані) і класифікує нові випадки більшістю голосів своїх K сусідів. При реалізації KNN (рис. 1.13), першим кроком є перетворення точок даних в їх математичні значення (вектори).

Алгоритм працює шляхом знаходження відстані між математичними значеннями цих точок. Він обчислює відстань між кожною точкою даних і тестовими даними, а потім знаходить ймовірність того, що точки будуть подібними до тестових даних. Класифікація заснована на тому, які точки входу поширюють найбільшу ймовірність.

Роботу KNN можна пояснити на основі наступного алгоритму:

1. Вибрати число K сусідів.
2. Обчислити евклідову відстань для K числа сусідів.
3. Взяти K найближчих сусідів відповідно до обчисленої евклідової відстані.
4. Серед цих K сусідів порахуйте кількість точок даних у кожній категорії.
5. Призначити нові точки даних до тієї категорії, для якої кількість сусідів максимальна.

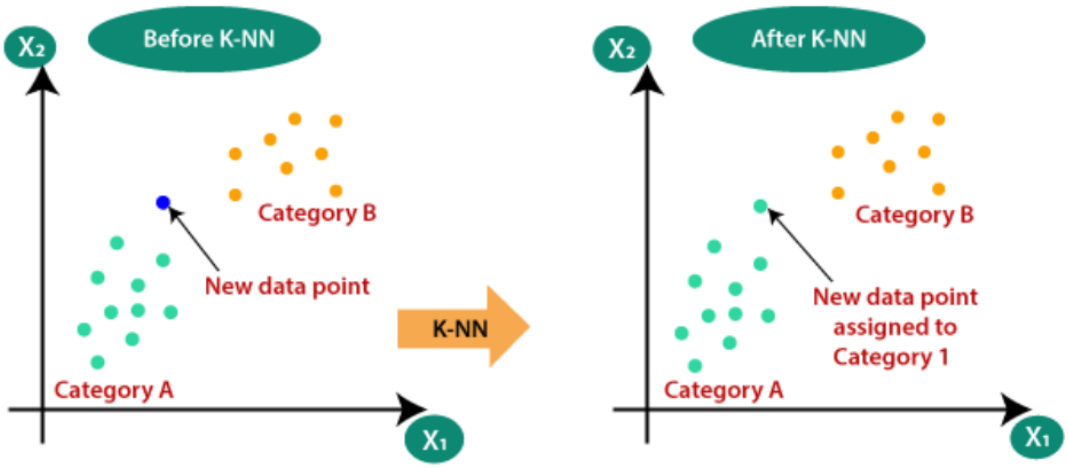


Рисунок 1.13 – Метод к-найближчих сусідів

Нажаль, немає заздалегідь визначених статистичних методів, щоб знайти найбільш сприятливе значення K. Але зазвичай притримуються таких правил:

1. Ініціалізувати випадкове значення K та почати обчислення.

2. Вибір невеликого значення K призводить до нестабільних меж прийняття рішень.
3. Значне значення K краще для класифікації, оскільки воно веде до згладжування меж рішення.
4. Вивести графік між частотою помилок і K, що позначає значення у визначеному діапазоні. Потім обрати значення K з мінімальною частотою помилок.

Отже, пам’ятаємо, що перший крок - обчислити відстань між новою точкою та кожною навчальною точкою. Функція відстані може бути Евклідовою, Мінковським (для безперервного) або Хеммінговою (для категоричного). [5]

Евклідова відстань:

$$d(x, x') = \sqrt{(x_1 - x'_1)^2 + \cdots + (x_n - x'_n)^2} \quad (1.10)$$

Відстань Мінковського: це відстань між реальними векторами з використанням суми їх абсолютної різниці.

$$d(x, x') = |(x_1 - x'_1)| + \cdots + |(x_n - x'_n)| \quad (1.11)$$

Відстань Хеммінга: використовується для категоріальних змінних. Якщо значення (x_n) і значення (x'_n) збігаються, відстань D буде дорівнює 0 . Інакше D=1.

$$d_H = \sum_{n=1}^{k}(x_n - x'_n), x = y \rightarrow d = 0, x \neq y \rightarrow d = 1 \quad (1.12)$$

Нарешті, вхід *x* призначається класу з найбільшою ймовірністю [6]:

$$P(y = j|X = x) = \frac{1}{K}\sum_{i \in A} I(y^{(i)} = j) \quad (1.13)$$

1.4 Висновки до розділу 1

Отже, у даному розділі було надано поняття основних об’єктів на яких базується дане дослідження. Вдалось ознайомитись із головними захворюваннями, які будуть класифікуватись у роботі.

Також було обрано кілька найефективніших методів для вирішення поставленої задачі. Детально розглянуто їх алгоритми та принципи роботи. Проте в кожному з них є певні недоліки, такі як розмірність, швидкість. В наступних розділах ці недоліки будуть частково усунені, що допоможе покращити результат.

2 МОДЕЛІ НЕЙРОННИХ МЕРЕЖ

2.1 Нейронні мережі у задачах класифікації

Нейронні мережі черпають натхнення з процесу навчання, що відбувається в мозку людини. Вони складаються зі штучної мережі функцій, які називаються параметрами, що дозволяє комп’ютеру навчатися та точно налаштовуватися, аналізуючи нові дані. Кожен параметр, який іноді також називають нейронами, є функцією, яка подає вихідні дані після отримання одного або кількох входів. Ці виходи потім передаються до наступного шару нейронів, які використовують їх як входи своєї власної функції і виробляють подальші виходи. І так продовжується, доки кожен шар нейронів не буде розглянуто, а кінцеві нейрони не отримають свій вхід. Ці кінцеві нейрони потім виводять остаточний результат для моделі.

На рис. 2.1 показано візуальне представлення такої мережі. Початковий вхід — це $\boldsymbol{x}$, який потім передається до першого шару нейронів (шари $\boldsymbol{h}$ на малюнку), де три функції розглядають вхідні дані, які вони отримують, і генерують вихідні. Потім цей результат передається до другого шару ($\boldsymbol{g}$ бульбашки на малюнку). Там подальший вихід розраховується на основі виходу з першого шару. Потім цей вторинний результат об’єднується, щоб отримати кінцевий результат моделі - $\boldsymbol{f}$.

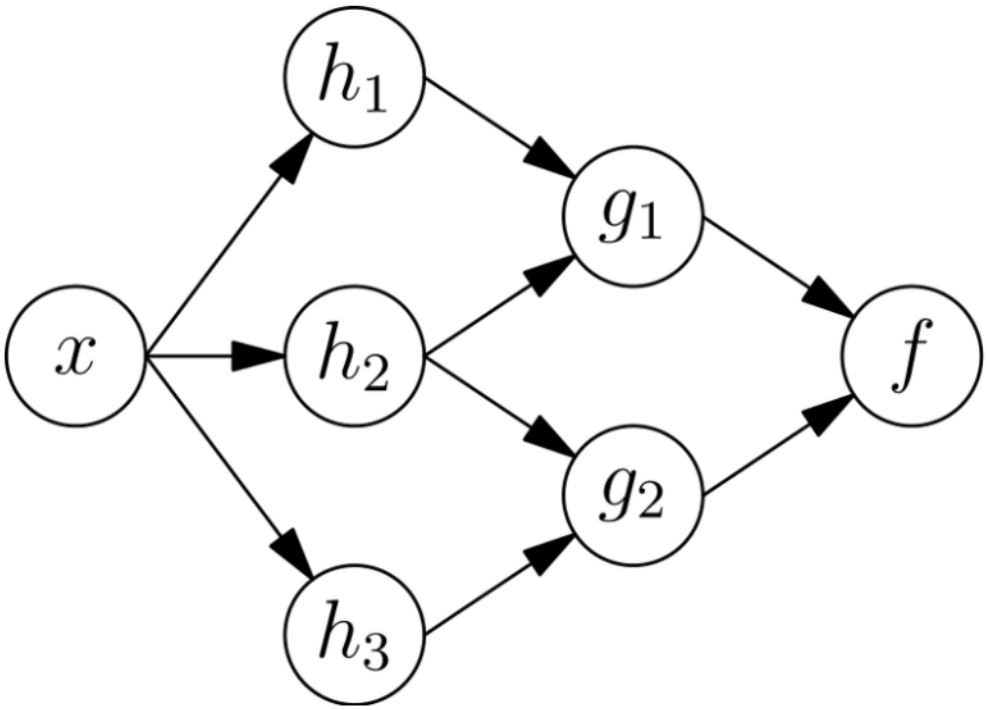


Рисунок 2.1 – Проста нейронна мережа

Альтернативний спосіб мислення про нейронну мережу — це думати про неї як про одну масивну функцію, яка приймає вхідні дані та отримує кінцевий вихід. Проміжні функції, які виконуються нейронами в їх багатьох шарах, зазвичай не спостерігаються і, на щастя, автоматизовані. Математика, що стоїть за ними, настільки ж цікава, як і складна, і заслуговує додаткового розгляду.

Як згадувалося раніше, нейрони в мережі взаємодіють з нейронами наступного шару, причому кожен вихід є входом для майбутньої функції. Кожна функція, включаючи початковий нейрон, отримує числовий вхід і виробляє числовий вихід на основі інтерналізованої функції, яка включає додавання члена зміщення, який є унікальним для кожного нейрона. Потім цей результат перетворюється на числовий вхід для функції наступного шару шляхом множення на відповідну вагу. Це продовжується до тих пір, поки не буде отримано один кінцевий вихід для мережі.

Складність полягає у визначенні оптимального значення для кожного члена зміщення, а також у знаходженні найкращого зваженого значення для

кожного проходу в нейронній мережі. Для цього необхідно вибрати функцію витрат. Функція витрат - це спосіб обчислення, наскільки конкретне рішення знаходиться на відстані від найкращого можливого рішення. Існує багато різних можливих функцій витрат, кожна з яких має переваги та недоліки, кожна з яких найкраще підходить за певних умов. Таким чином, функція вартості повинна бути адаптована та вибрана на основі індивідуальних потреб дослідження. Після визначення функції витрат нейронну мережу можна змінити таким чином, щоб мінімізувати цю функцію витрат.

Таким чином, простий спосіб оптимізації ваг і зміщення - це просто запустити мережу кілька разів. З першої спроби прогнози будуть випадковими. Після кожної ітерації буде проаналізована функція вартості, щоб визначити, як працює модель і як її можна покращити. Інформація, отримана від функції вартості, потім передається до функції оптимізації, яка обчислює нові значення ваги, а також нові значення зміщення. З цими новими значеннями, інтегрованими в модель, модель повторюється. Так триває, поки жодна зміна не покращить функцію витрат [8].

Існує три методи навчання: навчання з учителем, навчання без учителя та навчання з підкріпленням. Найпростішою з цих парадигм навчання є навчання з учителем, де нейронна мережа отримує позначені вхідні дані. Позначені приклади потім використовуються для висновку узагальнюючих правил, які можна застосувати до випадків без міток. Це найпростіший метод навчання, оскільки його можна розглядати як роботу з «вчителем» у формі функції, яка дозволяє мережі порівнювати свої прогнози з істинними та бажаними результатами. Саме цей метод і будемо використовувати у даному дослідженні. Методи без учителя не вимагають позначених початкових входів, а скоріше виводять правила та функції, засновані не тільки на даних, а й на виході мережі. Це перешкоджає тому типу прогнозів, які можна зробити. Замість того, щоб мати можливість класифікувати, така модель обмежується кластеризацією.

Усі завдання класифікації залежать від позначених наборів даних; тобто люди повинні передати свої знання в набір даних, щоб нейронна мережа дізналася кореляцію між мітками та даними. Нейронна мережа може розв’язати такі задачі класифікації:

1. Визначати обличчя, ідентифікувати людей на зображеннях, розпізнавати вирази обличчя (сердитий, радісний).
2. Визначати об’єкти на зображеннях (знаки зупинки, пішоходи, маркери смуг руху…).
3. Розпізнавати жести у відео.
4. Визначати голоси, ідентифікувати мовців, транскрибувати мовлення в текст, розпізнавати почуття в голосах.
5. Класифікувати текст як спам (в електронних листах) або шахрайський (у страхових виплатах); розпізнавати настрої в тексті (відгуки клієнтів).
6. Будь-які мітки, які можуть створювати люди, будь-які результати, які вас цікавлять і які корелюють з даними, можуть бути використані для навчання нейронної мережі [9].

2.2 Глибоке навчання

Мережі глибокого навчання відрізняються від більш поширених нейронних мереж з одним прихованим шаром; тобто кількість шарів вузлів, через які дані повинні проходити в багатоетапному процесі розпізнавання образів.

Попередні версії нейронних мереж, такі як перші персептрони, були неглибокими, складалися з одного вхідного та одного вихідного шарів і щонайбільше одного прихованого шару між ними. Більше трьох рівнів (включаючи вхідний і вихідний) кваліфікується як «глибоке» навчання.

У мережах глибокого навчання кожен рівень вузлів навчається окремому набору функцій на основі результатів попереднього рівня. Чим далі ви просуваєтеся в нейронну мережу, тим складніші функції можуть розпізнати ваші вузли, оскільки вони об’єднують і рекомбінують функції з попереднього шару (рис. 2.2).

Ієрархія особливостей, або відоме як ієрархія функцій, і це ієрархія зростаючої складності та абстракції. Це робить мережі глибокого навчання здатними обробляти дуже великі масиви даних з мільярдами параметрів, які проходять через нелінійні функції.

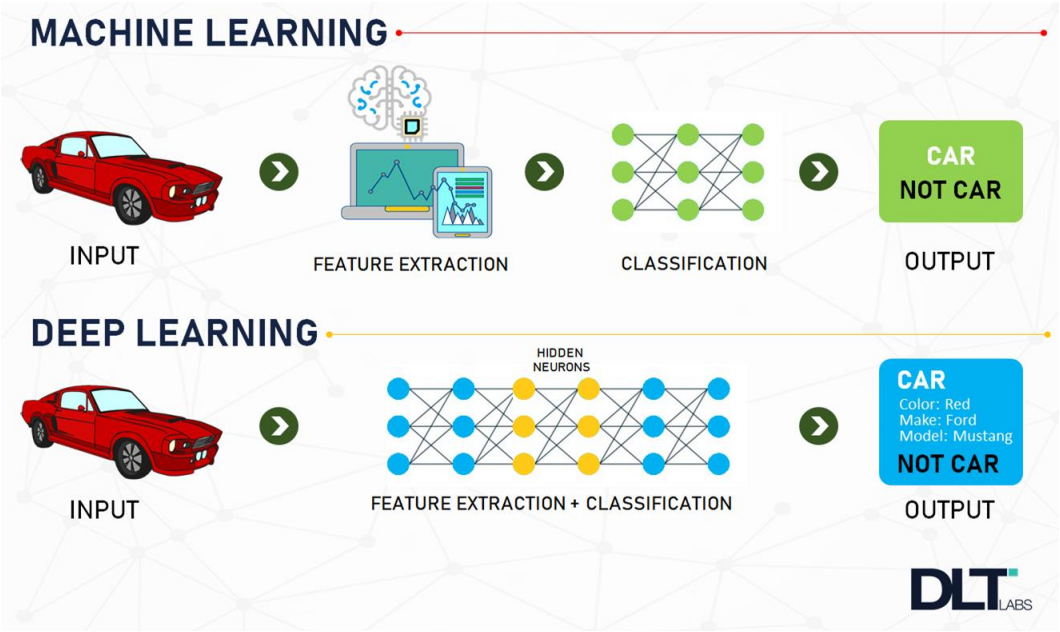


Рисунок 2.2 – Машинне і глибоке навчання

Перш за все, ці нейронні мережі здатні виявляти приховані структури в немаркованих, неструктурованих даних, які є переважною більшістю даних у світі. Іншим словом для неструктурованих даних є необроблені медіа; тобто зображення, тексти, відео та аудіозаписи. Таким чином, одна з проблем, які глибоке навчання вирішує найкраще, — це обробка та групування світових необроблених медіа без міток, виявлення подібності та аномалій у даних, які жодна людина не організувала у реляційній базі даних чи коли-небудь не назвала.

Наприклад, глибоке навчання може отримати мільйон зображень і згрупувати їх відповідно до їх схожості: кішки в одному кутку, криголами в іншому, а в третьому всі фотографії вашої бабусі. Це основа так званих розумних фотоальбомів.

З часовими рядами дані можуть групуватися навколо нормальної/здорової поведінки та аномальної/небезпечної поведінки. Якщо дані часового ряду генеруються за допомогою смартфона, це дасть уявлення про здоров'я та звички користувачів; якщо його генерує автозапчастина, його можна використовувати для запобігання катастрофічних поломок.

На відміну від більшості традиційних алгоритмів машинного навчання, мережі глибокого навчання виконують автоматичне вилучення функцій без втручання людини. Враховуючи, що вилучення функцій - це завдання, на виконання якого вчені -дослідники даних можуть витратити роки, глибоке навчання - це спосіб обійти завалу обмежених експертів. Це збільшує повноваження невеликих команд науки про дані, які за своєю природою не масштабуються.

Під час навчання на немаркованих даних кожен рівень вузла в глибокій мережі автоматично вивчає особливості, багаторазово намагаючись відновити вхідні дані, з яких він бере свої вибірки, намагаючись мінімізувати різницю між припущеннями мережі та розподілом ймовірностей самих вхідних даних. Наприклад, обмежені машини Больцмана створюють так звані реконструкції.

У процесі ці нейронні мережі вчаться розпізнавати кореляції між певними релевантними ознаками та оптимальними результатами – вони створюють зв'язки між сигналами ознак і тим, що ці функції представляють, чи то повна реконструкція, чи з позначеними даними.

Тоді мережа глибокого навчання, навчена міченим даним, може бути застосована до неструктурованих даних, надаючи їй доступ до набагато більше вхідних даних, ніж мережі машинного навчання. Це рецепт вищої продуктивності: чим більше даних може тренуватися мережа, тим точнішою вона буде. (Погані алгоритми, навчені великою кількістю даних, можуть

перевершувати хороші алгоритми, навчені дуже мало.) Здатність глибокого навчання обробляти та вчитися на основі величезної кількості немаркованих даних дає йому явну перевагу перед попередніми алгоритмами.

Мережі глибокого навчання закінчуються вихідним рівнем: логістичним або softmax-класифікатором, який призначає ймовірність певному результату або мітці. Ми називаємо це прогнозуванням, але це передбачення в широкому сенсі. Враховуючи вихідні дані у вигляді зображення, мережа глибокого навчання може вирішити, наприклад, що вхідні дані на 90 відсотків представляють людину [10].

2.3 Згорткова нейронна мережа

Різновидом нейронної мережі є згорткова нейронна мережа. ConvNet, як їх іноді називають, пропонують деякі суттєві переваги перед звичайними нейронними мережами, особливо коли мова йде про класифікацію зображень. У такому випадку початковими вхідними даними будуть зображення, що складаються з пікселів. Традиційна проблема з класифікацією зображень полягає в тому, що з великими зображеннями, з багатьма кольоровими каналами, обчислення стають нездійсненним для навчання деякі моделі. Те, що CNN намагається зробити, це перетворити зображення у форму, яку легше обробляти, зберігаючи при цьому найважливіші характеристики. Це робиться шляхом проходження фільтра над початковим зображенням, яке проводить множення матриці на підрозділ пікселів у початковому зображенні, воно повторює підмножини, поки не розгляне всі підмножини. Фільтр має на меті захопити найважливіші функції, одночасно дозволяючи усунути зайві функції. Таке проходження фільтра над початковими пікселями відоме як шар згортки.

Згортковий шар є основним будівельним блоком CNN, і саме там відбувається більшість обчислень. Для цього потрібно кілька компонентів, а

саме вхідні дані, фільтр і карта об’єктів. Припустимо, що вхідним буде кольорове зображення, яке складається з матриці пікселів у 3D. Це означає, що вхідні дані матимуть три виміри - висоту, ширину та глибину - які відповідають RGB на зображенні. У нас також є детектор функцій, також відомий як ядро або фільтр, який буде переміщатися по сприйнятливих полях зображення, перевіряючи, чи є функція. Цей процес відомий як згортка (рис. 2.3).

Детектор ознак — це двовимірний (2-D) масив ваг, який представляє частину зображення. Хоча вони можуть відрізнятися за розміром, розмір фільтра зазвичай становить матрицю 3x3; це також визначає розмір рецептивного поля. Потім фільтр застосовується до області зображення, і між вхідними пікселями і фільтром обчислюється крапковий добуток. Цей точковий добуток потім подається у вихідний масив. Після цього фільтр зміщується на один крок, повторюючи процес, поки ядро не охопить усе зображення. Кінцевий вихід із ряду точкових добутків із входу та фільтра відомий як карта об’єктів, карта активації або згорнута функція .

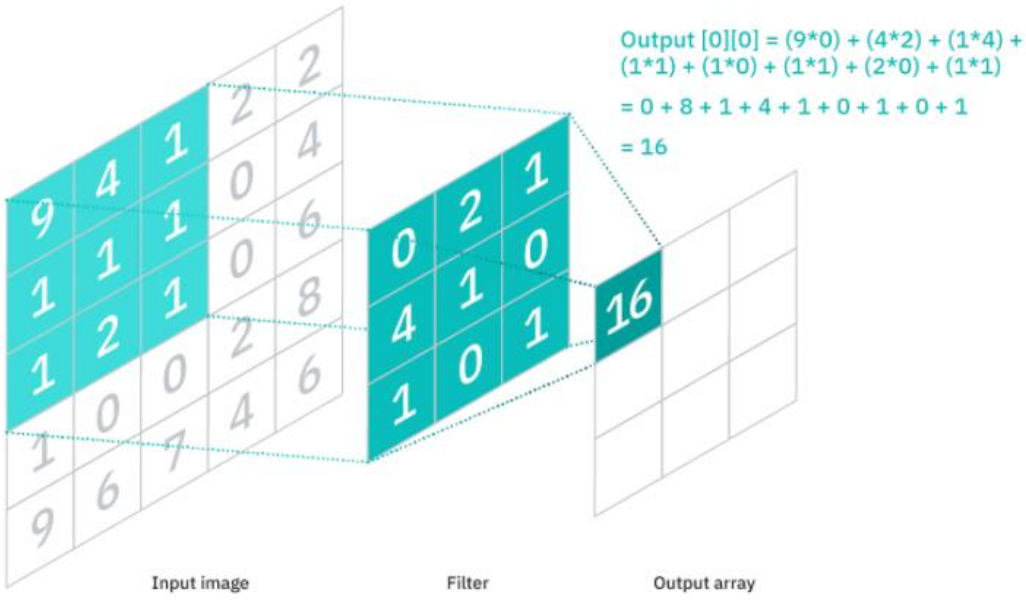


Рисунок 2.3 – Шар згортки

Як можна побачити на рисунку 2.3, кожне вихідне значення на карті об’єктів не обов’язково має з’єднуватися з кожним значенням пікселя у вхідному зображенні. Потрібно лише підключитися до сприйнятливого поля, де застосовується фільтр. Оскільки вихідному масиву не потрібно відображати безпосередньо на кожне вхідне значення, згорткові (і об'єднуючі) шари зазвичай називають «частково з'єднаними» шарами. Однак цю характеристику можна також описати як локальну зв’язність.

Потрібно зауважити, що ваги в детекторі функцій залишаються фіксованими, коли він рухається по зображенню, що також відоме як спільне використання параметрів. Деякі параметри, як-от значення вагів, коригуються під час навчання за допомогою процесу зворотного поширення та градієнтного спуску. Однак існують три гіперпараметри, які впливають на обсяг виводу, який необхідно встановити перед початком навчання нейронної мережі. До них належать:

1. Кількість фільтрів впливає на глибину виводу. Наприклад, три різні фільтри дадуть три різні карти характеристик, створюючи глибину до трьох.
2. Stride – це відстань або кількість пікселів, на які ядро переміщається по вхідній матриці. Хоча значення кроку два або більше зустрічається рідко, більший крок дає гірший результат.
3. Zero-padding зазвичай використовується, коли фільтри не підходять до вхідного зображення. Це встановлює всі елементи, які виходять за межі вхідної матриці, на нуль, створюючи вихід більшого або такого ж розміру. Існує три типи підкладки:
    a. Valid padding: це також відоме як відсутність заповнення. У цьому випадку остання згортка відкидається, якщо розміри не вирівнюються.
    б. Same padding: це заповнення гарантує, що вихідний шар має той самий розмір, що і вхідний шар

в. Full padding: цей тип заповнення збільшує розмір виводу, додаючи нулі до межі вхідних даних.

Після шару згортки йде шар об’єднання, який буде намагатися зменшити просторовий розмір згорнутих об’єктів. Зменшення складності, яке іноді називають зменшенням розмірності, зменшить обчислювальні витрати на виконання аналізу набору даних, дозволяючи методу бути більш надійним. У цьому шарі ядро знову проходить через усі підмножини пікселів зображення. Є два типи ядер об’єднання, які зазвичай використовуються. Перший - це максимальне об'єднання (max pooling), яке зберігає максимальне значення підмножини. Альтернативним ядром є середнє об’єднання (average pooling), яке робить саме те, що ви очікували: воно зберігає середнє значення всіх пікселів у підмножині (рис 2.4).

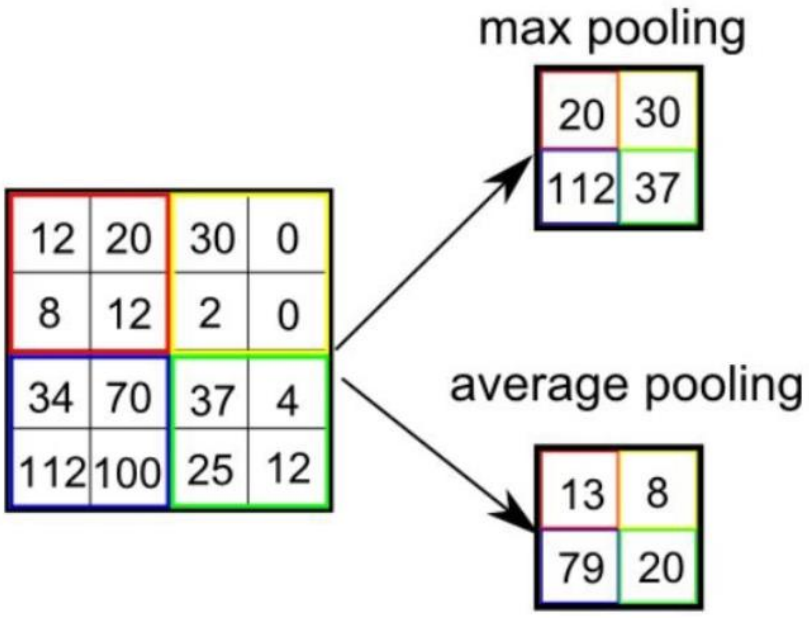


Рисунок 2.4 – Pooling шари

Далі йде шар під назвою fully-connected (повнозв’язний). Як згадувалося раніше, значення пікселів вхідного зображення не пов’язані безпосередньо з вихідним шаром у частково з’єднаних шарах. Однак у цьому шарі кожен вузол вихідного шару підключається безпосередньо до вузла попереднього шару.

Цей шар виконує завдання класифікації на основі ознак, витягнутих через попередні шари та їх різні фільтри. У той час як рівні згортки та об’єднання зазвичай використовують функції ReLU, на рівні fully-connected зазвичай використовують функцію активації softmax для належної класифікації вхідних даних, створюючи ймовірність від 0 до 1.

Отже, на виході наша згорткова нейронна мережа буде виглядати таким чином (рис. 2.5) [11]:

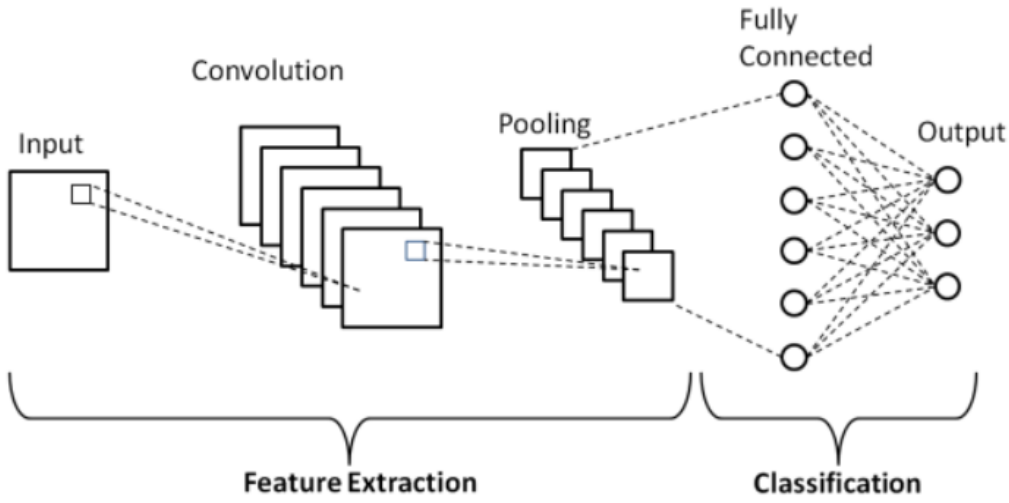


Рисунок 2.5 – Згорткова нейронна мережа

2.4 Функції ReLu та Softmax для побудови оптимальної архітектури

Функція ***ReLU*** (рис. 2.6) проста і не складається з важких обчислень, оскільки немає складної математики.

$$f(x) = \max(0, x) \tag{2.1}$$

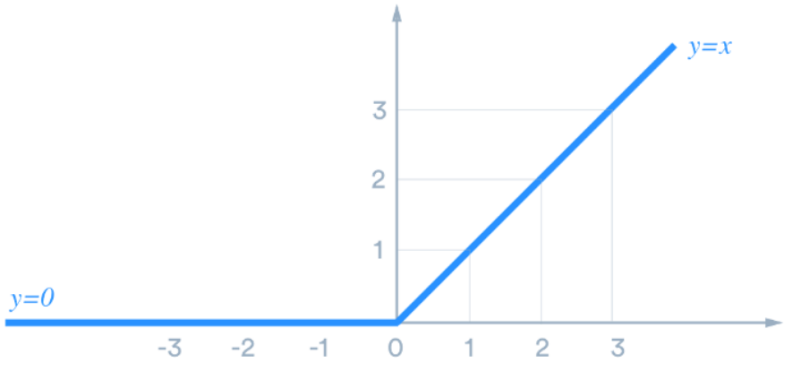


Рисунок 2.6 – Графік функції ReLU

Таким чином, моделі може знадобитися менше часу на тренування. Ще однією важливою властивістю, яку ми вважаємо перевагою використання функції активації ReLU, є розрідженість.

Зазвичай матриця, в якій більшість записів дорівнює 0, називається розрідженою матрицею, і аналогічним чином ми бажаємо мати подібну властивість у наших нейронних мережах, де деякі вагові показники дорівнюють нулю. Розрідженість призводить до стислих моделей, які часто мають кращий передбачуваний результат та менше шуму. У розрідженій мережі більш ймовірно, що нейрони насправді обробляють значущі аспекти проблеми. Наприклад, у моделі, яка виявляє людські обличчя на зображеннях, може існувати нейрон, який може ідентифікувати вуха, який, очевидно, не слід активувати, якщо зображення не є обличчям, а є кораблем або горою.

Оскільки ReLU дає нульовий вихід для всіх негативних входів, ймовірно, що будь-який даний блок взагалі не активується, що спричиняє розрідженість мережі. Тепер проаналізуємо, наскільки функція активації ReLu краща, ніж раніше відомі функції активації, і чому слід обрати саме її.

Функції активації, які використовувалися здебільшого до ReLU, такі як сигмоїдні або tanh функція активації, насичені. Це означає, що великі значення прив’язуються до 1, 0, а малі – до -1 або 0 для tanh і sigmoid відповідно. Крім того, функції дійсно чутливі до змін лише в середині їхнього входу, наприклад,

0,5 для сигмоїдна і 0,0 для tanh. Через це у них виникла проблема, яка називається проблемою зникаючого градієнта.

Але звичайно у ReLU є свої недоліки, наприклад, вибухаючий градієнт. Градієнт, що вибухає, протилежний градієнту, що зникає, і виникає там, де накопичуються великі градієнти помилок і призводять до дуже великих оновлень ваг моделі нейронної мережі під час навчання. Через це модель нестабільна і не може вчитися з навчальних даних.

Крім того, існує зворотна сторона нульового значення для всіх негативних значень, і ця проблема називається «вмираючим ReLU». Нейрон ReLU «мертвий», якщо він застряг у негативній стороні і завжди виводить 0. Оскільки нахил ReLU в мінусі діапазон також дорівнює 0, як тільки нейрон стає негативним, навряд чи він відновиться. Такі нейрони не відіграють жодної ролі в розрізненні вхідних даних і по суті є марними. Згодом велика частина вашої мережі може нічого не робити. Проблема вмирання, ймовірно, виникне, коли швидкість навчання занадто висока або є велике негативне упередження [12].

Функція ***Softmax*** – це функція, яка перетворює вектор із K реальних значень у вектор із K реальних значень, які у сумі дають 1 (рис. 2.7). Вхідні значення можуть бути додатними, від’ємними, нульовими або більшими за одиницю, але softmax перетворює їх у значення між 0 і 1, щоб їх можна було інтерпретувати як ймовірності. Якщо один із вхідних даних є малим або негативним, softmax перетворює його на малу ймовірність, а якщо вхід великий, то перетворює його на велику ймовірність, але вона завжди залишатиметься між 0 та 1.

$$\sigma\left(\underset{Z}{\rightarrow}\right)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_i}}, \tag{2.2}$$

де z_i - вхідний вектор, K – кількість класів.

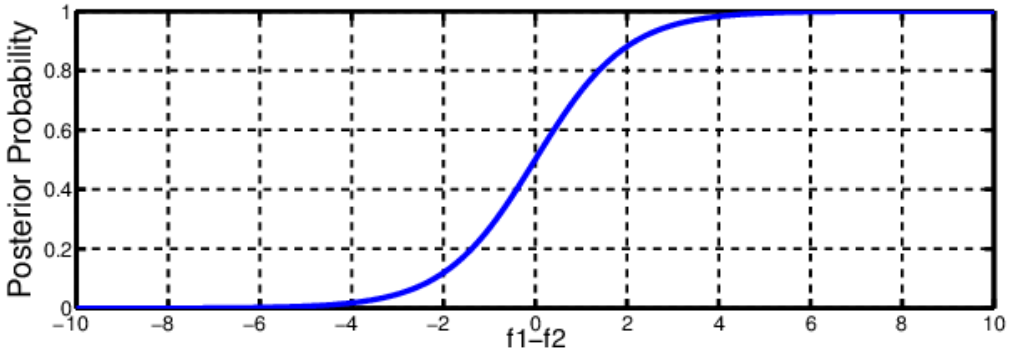


Рисунок 2.7 – Графік функції Softmax

Функцію softmax іноді називають функцією softargmax або багатокласовою логістичною регресією. Це пояснюється тим, що softmax є узагальненням логістичної регресії, яку можна використовувати для багатокласової класифікації, а її формула дуже схожа на сигмоїдну функцію, яка використовується для логістичної регресії. Функцію softmax можна використовувати в класифікаторі лише тоді, коли класи є взаємовиключними.

Більшість багатошарових нейронних мереж закінчуються передостаннім шаром, який виводить оцінки з реальними значеннями, які не зручно масштабуються і з якими може бути важко працювати. Тут дуже корисний softmax, оскільки він перетворює оцінки в нормований розподіл ймовірностей, який можна відобразити користувачеві або використати як вхідні дані для інших систем. З цієї причини зазвичай додається функція softmax як кінцевий шар нейронної мережі [13].

2.5 Аналіз сучасних нейронних мереж

EfficientNet - клас нових моделей, який вийшов з вивчення масштабування (скейлінг, scaling) моделей та балансування між собою глибини та ширини (кількості каналів) мережі, а також дозволу зображень у мережі. Автори мережі пропонують новий метод складового масштабування

(compound scaling method), який рівномірно масштабує глибину/ширину/дозвіл із фіксованими пропорціями між ними. З існуючого методу під назвою Neural Architecture Search для автоматичного створення нових мереж та свого власного методу масштабування автори отримують новий клас моделей під назвою EfficientNets (рис. 2.8) [14].

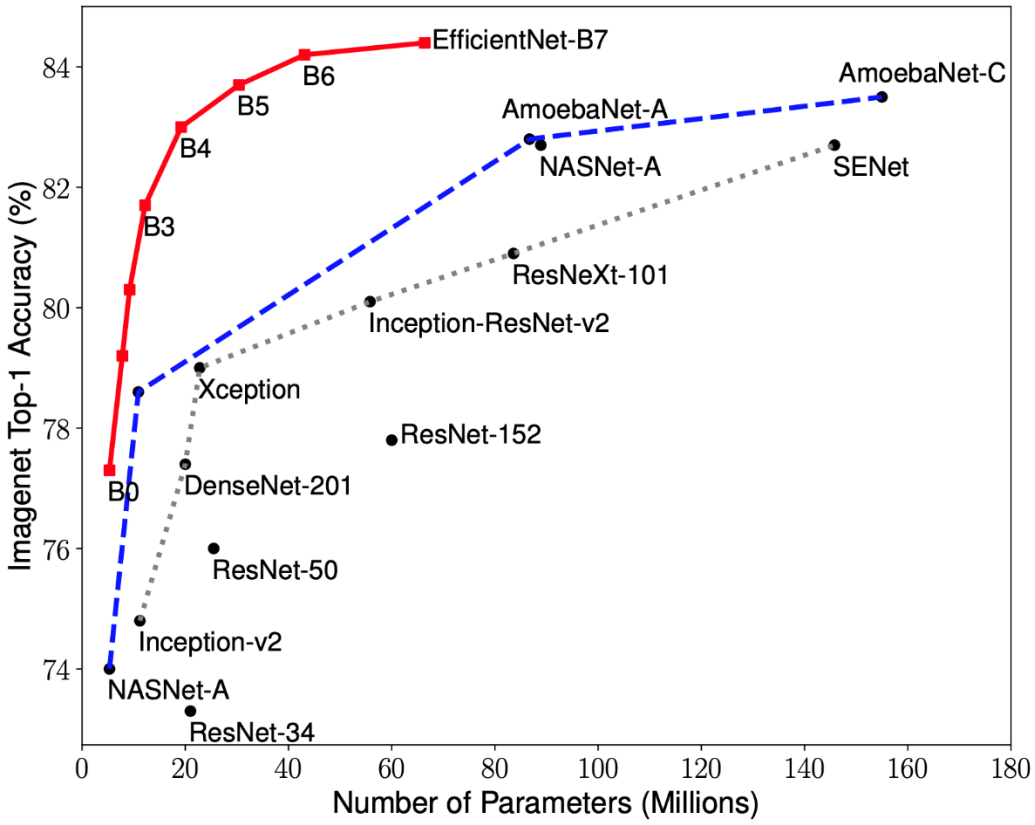


Рисунок 2.8 – Залежність точності від кількості параметрів

VGG (Visual Geometry Group) – це стандартна архітектура глибокої згорткової нейронної мережі (CNN) з кількома шарами. «Глибокий» відноситься до кількості шарів з VGG-16 або VGG-19, що складається з 16 і 19 згорткових шарів.

Архітектура VGG є основою інноваційних моделей розпізнавання об'єктів. Розроблена як глибока нейронна мережа, VGGNet також перевершує базові лінії для багатьох завдань і наборів даних за межами ImageNet. Більше того, зараз це одна з найпопулярніших архітектур розпізнавання зображень. (рисунок 2.9).

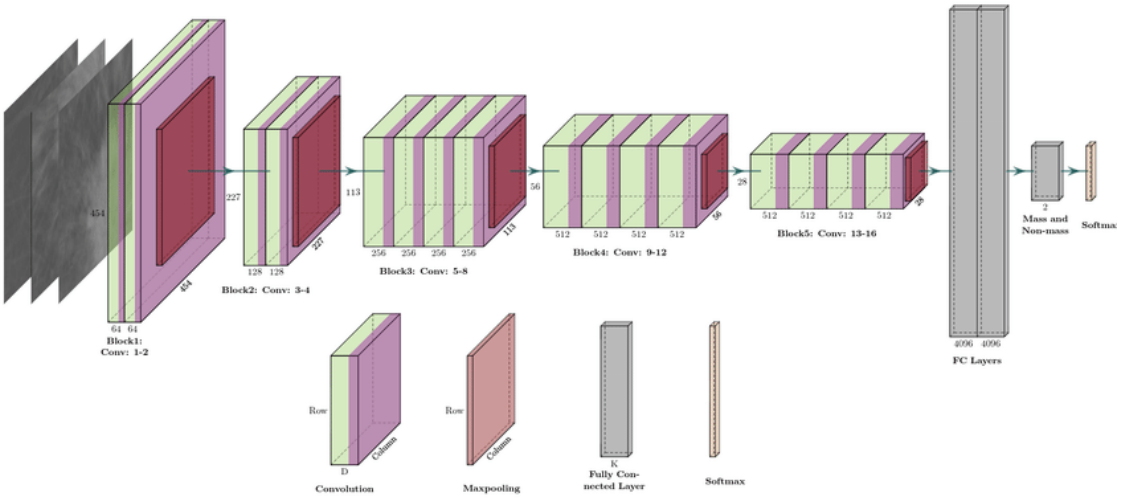


Рисунок 2.9 – Архітектура VGG

VGGNets засновані на найважливіших характеристиках згорткових нейронних мереж (CNN). Мережа VGG побудована з дуже малими згортковими фільтрами. VGG-16 складається з 13 згорткових шарів і трьох повністю пов’язаних шарів.

Розглянемо архітектуру VGG:

1. Вхід: VGGNet приймає зображення розміром 224×224. Для конкурсу ImageNet творці моделі вирізали центральний латок розміром 224×224 на кожному зображенні, щоб зберегти відповідний розмір вхідного зображення.
2. Згорткові шари: згорткові шари VGG використовують мінімальне сприйнятливе поле, тобто 3×3, найменший можливий розмір, який все ще захоплює вгору/вниз і вліво/вправо. (рис. 2.10) Крім того, існують також фільтри згортки 1×1, які діють як лінійне перетворення входу. Далі йде модуль ReLU, який є величезною інновацією від AlexNet, що скорочує час навчання. ReLU означає функцію активації випрямленого лінійного блоку; це кусково-лінійна функція, яка виведе вхідні дані, якщо вони позитивні; інакше вихід дорівнює нулю. Крок згортки фіксується на рівні 1 піксель, щоб зберегти просторову роздільну здатність після

згортки (крок — це кількість зсувів пікселя над вхідною матрицею).

3. Приховані шари: усі приховані шари в мережі VGG використовують ReLU. VGG зазвичай не використовує локальну нормалізацію відповіді (LRN), оскільки збільшує споживання пам’яті та час навчання. Більше того, це не покращує загальну точність.

Повністю підключені шари: VGGNet має три повністю підключені шари. З трьох шарів перші два мають по 4096 каналів кожен, а третій має 1000 каналів, по 1 для кожного класу.

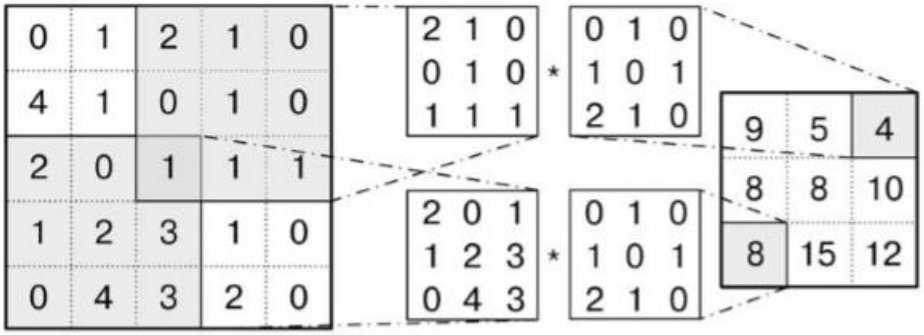


Рисунок 2.10 – Приклад згортки з ядром 3x3 та кроком 1 піксель

Кількість фільтрів, які ми можемо використовувати, подвоюється на кожному кроці або через кожен стек шару згортки. Це основний принцип, який використовується для розробки архітектури мережі VGG. Одним із важливих недоліків мережі VGG16 є те, що це величезна мережа, а це означає, що для навчання її параметрів потрібно більше часу.

Завдяки глибині та кількості повністю підключених шарів, модель VGG16 становить понад 533 МБ. Це робить впровадження мережі VGG трудомістким завданням.

Модель VGG16 використовується в кількох проблемах класифікації зображень глибокого навчання, але менші архітектури мережі, такі як GoogLeNet і SqueezeNet, часто є кращими. У будь-якому випадку, VGGNet є

чудовим будівельним блоком для цілей навчання, оскільки його легко реалізувати [15].

VGG означає Visual Geometry Group і складається з блоків, де кожен блок складається з шарів 2D Convolution і Max Pooling. Випускається в двох моделях — VGG16 і VGG19 — з 16 і 19 шарами.

Зі збільшенням кількості шарів у CNN збільшується здатність моделі відповідати більш складним функціям. Отже, більше шарів обіцяє кращу продуктивність. Це не слід плутати зі штучною нейронною мережею (ANN), де збільшення кількості шарів не обов’язково призводить до кращої продуктивності.

Тепер виникає питання, чому б вам не використовувати VGGNet з більшою кількістю шарів, наприклад VGG20, або VGG50, або VGG100? Ось тут і виникає проблема. Вагові коефіцієнти нейронної мережі оновлюються за допомогою алгоритму зворотного поширення, який вносить незначні зміни до кожної ваги, щоб зменшити втрати моделі. Він оновлює кожну вагу, щоб зробити крок у напрямку зменшення втрати. Це не що інше, як градієнт цієї ваги, який можна знайти за допомогою правила ланцюга.

Однак, оскільки градієнт продовжує рухатися назад до початкових шарів, значення продовжує збільшуватися з кожним локальним градієнтом. Це призводить до того, що градієнт стає все меншим і меншим, що робить зміни початкових шарів дуже малими. Це, в свою чергу, значно збільшує час навчання. Проблему можна вирішити, якщо локальний градієнт стає рівним 1. Ось тут і входить ResNet, оскільки він досягає цього за допомогою функції ідентифікації. Отже, оскільки градієнт поширюється назад, його значення не зменшується, оскільки локальний градієнт дорівнює 1.

Глибокі залишкові мережі (ResNets), такі як популярна модель ResNet-50, є ще одним типом архітектури згорткової нейронної мережі (CNN), яка має глибину 50 шарів. Залишкова нейронна мережа використовує вставку скорочених з'єднань для перетворення звичайної мережі в її аналогову

залишкову мережу. У порівнянні з VGGNets, ResNets менш складні, оскільки мають менше фільтрів.

ResNet, також званий залишковою мережею, не дозволяє виникати проблемі зникаючого градієнта. Пропускні сполучення (рисунок 2.11) діють як градієнтні супермагістралі, які дозволяють градієнту протікати без перешкод. Це також одна з найважливіших причин, чому ResNet поставляється у таких версіях, як ResNet50, ResNet101 і ResNet152.

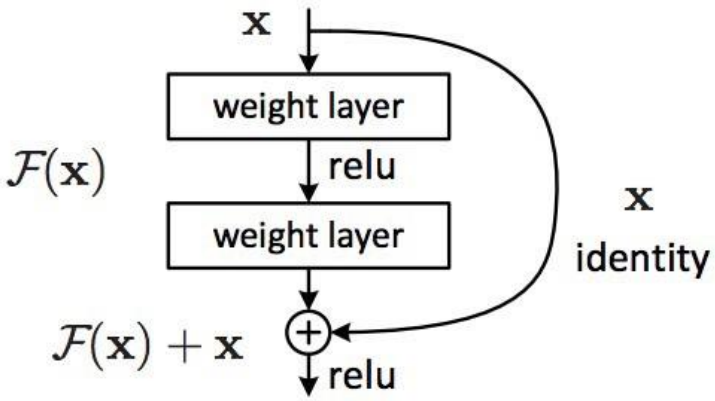


Рисунок 2.11 – Особливість ResNet

ResNet з великою кількістю шарів почав використовувати шар схожий на «вузьке місце», як у модуля Inception (рисунок 2.12):

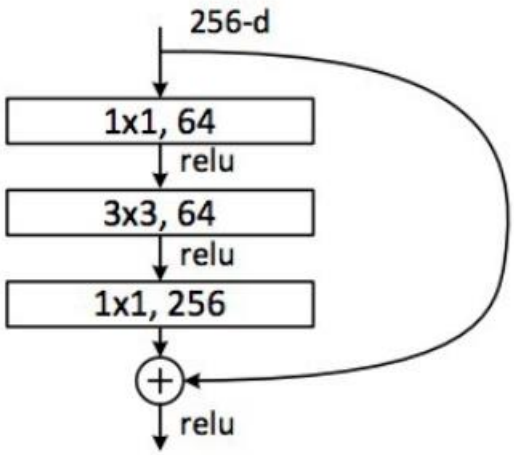


Рисунок 2.12 – Шар «вузького місця» для ResNet

Підводячи підсумок, залишкова мережа або ResNet була головною інновацією, яка змінила навчання глибоких згорткових нейронних мереж для завдань, пов’язаних з комп’ютерним зором. У той час як оригінальний Resnet мав 34 шари і використовував 2-шарові блоки, інші вдосконалені варіанти, такі як Resnet50, використовували 3-шарові блоки вузьких місць, щоб забезпечити покращену точність і менший час навчання [16].

2.6 Висновки до розділу 2

Отже, у цьому розділі були розглянуті найпопулярніші методи навчання нейронних мереж та обрано найоптимальніші для даного дослідження. Також проведено поверхневий аналіз та порівняння сучасних архітектур.

3 ПОБУДОВА КЛАСИФІКАТОРА ДЛЯ ЗОБРАЖЕНЬ МАНІОКИ. АНАЛІЗ РЕЗУЛЬТАТІВ

Основним завданням цієї роботи та проекту було створення класифікатора для зображень маніоки, який зможе розпізнавати чотири категорії хвороб або вказувати на здоровий лист. За допомогою цього фермери зможуть швидко виявити хворі рослини, потенційно врятуючи їхні врожаї, перш ніж вони завдадуть непоправної шкоди.

Робота проводилась у декілька етапів:

1) підбір методів машинного навчання для класифікації;
2) підготовка даних;
3) обробка даних;
4) реалізація методів;
5) запуск програми на підготовлених даних;
6) збір та аналіз результатів.

3.1 Опис вхідних даних

У цій роботі представлений набір даних із 21 367 маркованих зображень, зібраних під час регулярного опитування в Уганді. Більшість зображень були отримані від фермерів, які фотографували свої сади, і анотовані експертами Національного інституту досліджень ресурсів сільськогосподарських культур

(NaCRRI) у співпраці з лабораторією ШІ в Університеті Макерере, Кампала. Це у форматі, який найбільш реалістично представляє те, що фермерам потрібно було б діагностувати в реальному житті.

У цій роботі буде запропоновано відрізнити кілька захворювань, які завдають матеріальної шкоди продовольству багатьох африканських країн. У деяких випадках основним засобом є спалення заражених рослин, щоб запобігти подальшому поширенню, що може зробити швидкий автоматизований перехід дуже корисним для фермерів.

Інформація зберігається у трьох папках та таких файлах:

1) test_images - папка з даними для тестування моделі;
2) train_images – папка з даними для тренування моделі (рис. 3.1);
3) train.csv – мічені дані для тренування (таблиця 3.1):
   а) image_id – назва зображення;
   б) label – ID захворювання;
4) sample_submission.csv – правильно відформатований зразок подання з урахуванням розкритого вмісту тестового набору:
   а) image_id – назва зображення;
   б) label – передбачуване ID захворювання;
5) label_num_to_disease_map.json – файл співставлення між кожним ID захворювання та назвою хвороби (рис. 3.2).

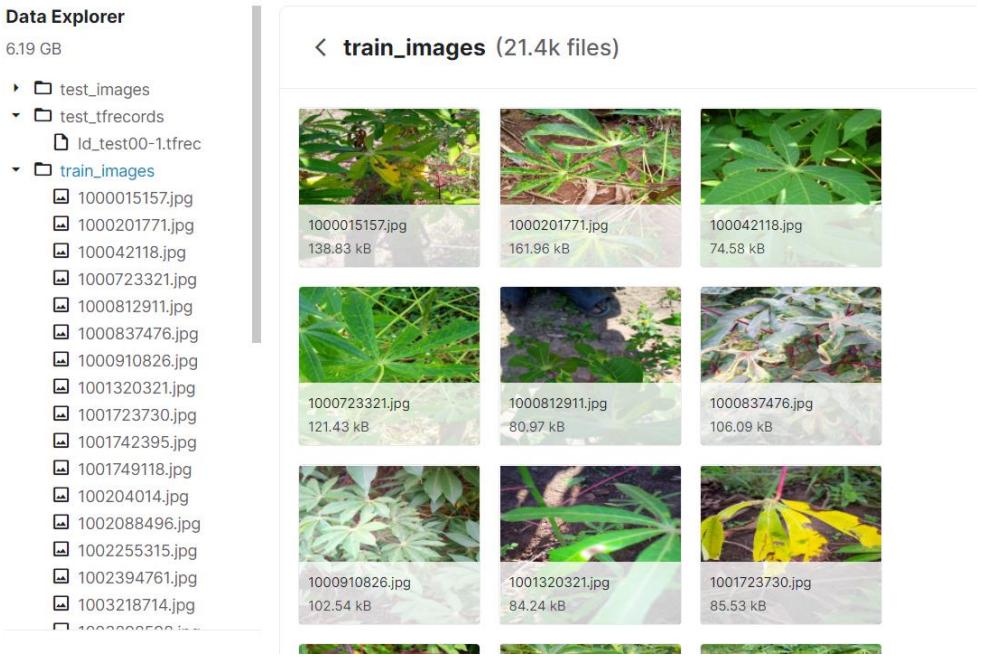


Рисунок 3.1 – Вхідні дані

Таблиця 3.1 – Вхідні тренувальні дані

№	image_id	label
1	1000015157.jpg	0
2	1000201771.jpg	3
3	100042118.jpg	1
…	……	…
21398	999998473.jpg	4

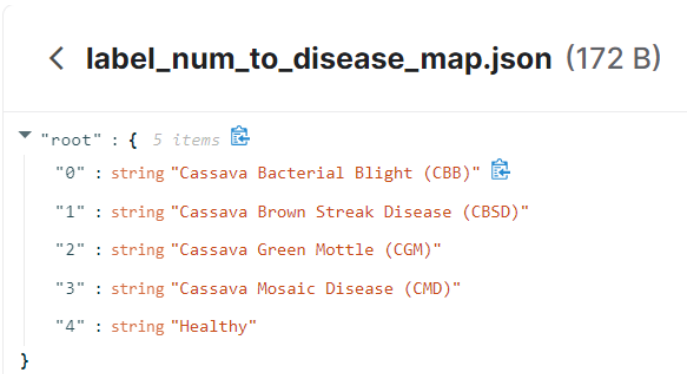


Рисунок 3.2 – Класифікація захворювань

3.2 Попередня обробка даних

Перед початком роботи програми дані потрібно проаналізувати та обробити. Завантажимо дані та переконаємось, що вони правильно зчитані (рис. 3.3).

```
train = pd.read_csv('../input/cassava-leaf-disease-classification/train.csv')
test = pd.read_csv('../input/cassava-leaf-disease-classification/sample_submission.csv')
label_map = pd.read_json('../input/cassava-leaf-disease-classification/label_num_to_disease_map.
json',
                         orient='index')
```

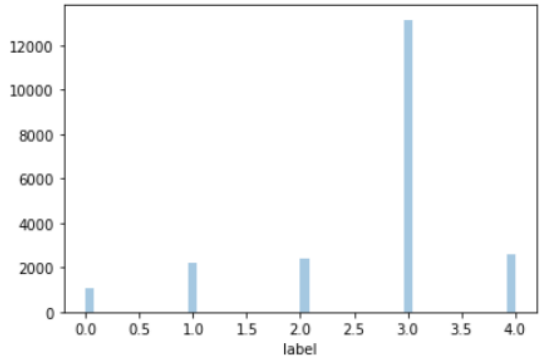
	image_id	label
0	1000015157.jpg	0
1	1000201771.jpg	3
2	100042118.jpg	1
3	1000723321.jpg	1
4	1000812911.jpg	3

	image_id	label
0	2216849948.jpg	4

	0
0	Cassava Bacterial Blight (CBB)
1	Cassava Brown Streak Disease (CBSD)
2	Cassava Green Mottle (CGM)
3	Cassava Mosaic Disease (CMD)
4	Healthy

Рисунок 3.3 – Завантаження даних

Тепер перевіримо як виглядають листя з хворобами кожного класу (або здорове листя) та яку частину вибірки вони представляють (рис.3.4, 3.5):

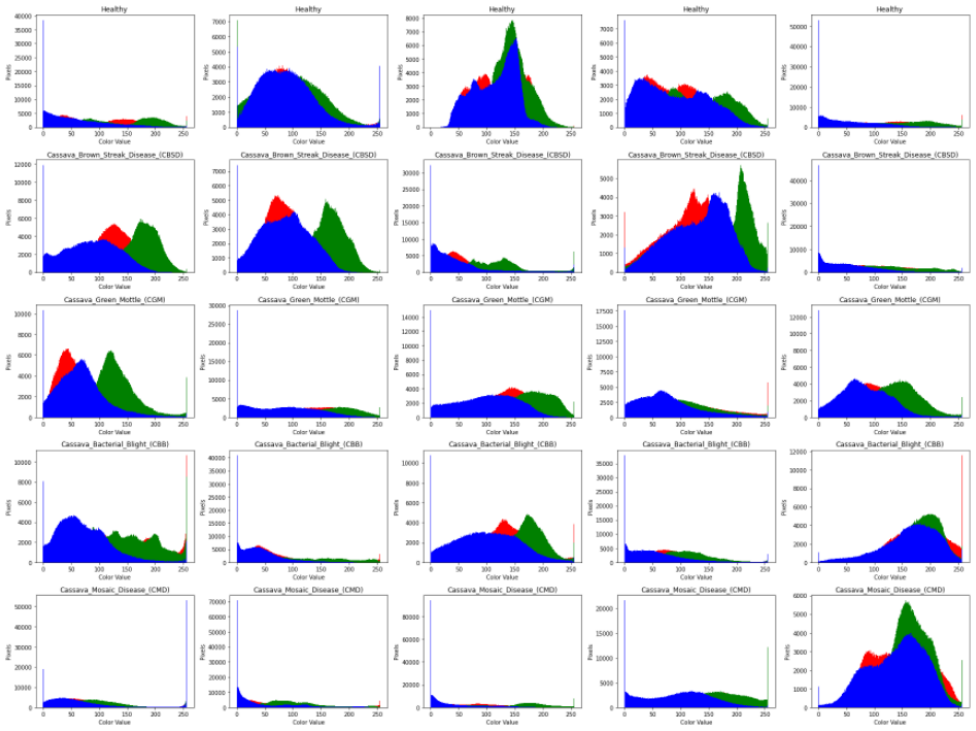



```
{'Healthy': 0.12,
 'Cassava_Brown_Streak_Disease_(CBSD)': 0.1,
 'Cassava_Green_Mottle_(CGM)': 0.11,
 'Cassava_Bacterial_Blight_(CBB)': 0.05,
 'Cassava_Mosaic_Disease_(CMD)': 0.62}
```

Рисунок 3.4 – Кількість представників

Рисунок 3.5 – Зображення листків з хворобами

Розглянемо ці зображення зі сторони кольорової моделі RGB. Це допоможе знайти залежність між кольором листя та певною хворобою, або ж спростувати існування даної кореляції (рис. 3.6).




```
{'Healthy': (109.8735570625, 124.46476510416666, 81.59887583333334),
 'Cassava_Brown_Streak_Disease_(CBSD)': (107.28399364583336,
  122.0959645625,
  76.03142877083333),
 'Cassava_Green_Mottle_(CGM)': (112.63699750000002,
  128.8406809375,
  84.21251499999998),
 'Cassava_Bacterial_Blight_(CBB)': (105.48185189583336,
  120.35613425,
  67.76706339583333),
 'Cassava_Mosaic_Disease_(CMD)': (112.9423095625,
  127.11148672916666,
  85.78937264583334)}
```

Рисунок 3.6 – RGB канали у зображеннях

Як видно з малюнку та результатів, залежність між кольорами та класами захворювань дуже низька. Тому проведемо деякі маніпуляції із зображеннями, що допоможе уникнути похибки за рахунок RGB-каналів та нормалізувати дані (рис. 3.7).

```python
def get_transforms(*, data):

    if data == 'train':
        return Compose([
            #Resize(CFG.size, CFG.size),
            RandomResizedCrop(CFG.size, CFG.size),
            Transpose(p=0.5),
            HorizontalFlip(p=0.5),
            VerticalFlip(p=0.5),
            ShiftScaleRotate(p=0.5),
            Normalize(
                mean=[0.485, 0.456, 0.406],
                std=[0.229, 0.224, 0.225],
            ),
            ToTensorV2(),
        ])

    elif data == 'valid':
        return Compose([
            Resize(CFG.size, CFG.size),
            Normalize(
                mean=[0.485, 0.456, 0.406],
                std=[0.229, 0.224, 0.225],
            ),
            ToTensorV2(),
        ])
```

```python
train_dataset = TrainDataset(train, transform=get_transforms(data='train'))

for i in range(1):
    image, label = train_dataset[i]
    plt.imshow(image[0])
    plt.title(f'label: {label}')
    plt.show()
```

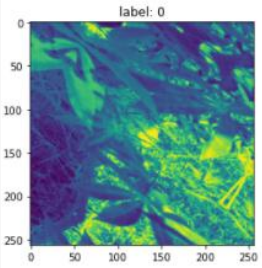


Рисунок 3.7 – Нормалізація даних

Було проведено огляд класів та зображень директорій train та validation. Результати показали, що test зберігає 17118 зображень, а validation 4279. Обидва випадки мають 5 класів, що було очікувано (Рис. 3.8).

```python
train_dataset = TrainDataset(train_folds,
                             transform=get_transforms(data='train'))
valid_dataset = TrainDataset(valid_folds,
                             transform=get_transforms(data='valid'))

train_loader = DataLoader(train_dataset,
                          batch_size=CFG.batch_size,
                          shuffle=True,
                          num_workers=CFG.num_workers, pin_memory=True, drop_last=True)
valid_loader = DataLoader(valid_dataset,
                          batch_size=CFG.batch_size,
                          shuffle=False,
                          num_workers=CFG.num_workers, pin_memory=True, drop_last=False)
```

Рисунок 3.8 – Огляд train та validation

3.3 Опис частин програми

Прикладна частина дослідження була проведена за допомогою мови програмування Python 3.9 та бібліотек Keras, Numpy та Pandas, оскільки такий інструментарій найкраще підходить для виконання поставленої задачі, а також для проектів машинного навчання та штучного інтелекту. Python надає доступ до бібліотек та фреймворків для машинного навчання (Machine Learning). Проекти на Python дуже легкі у своїй реалізації, та можуть бути запущені на будь-якому комп’ютері або на будь-якій сьогоднішній операційній системі.

Бібліотека Keras містить численні втілення широко вживаних нейромережних будівельних блоків, таких як шари, цільові та передавальні функції, оптимізувальники та безліч інструментів для спрощення роботи із зображеннями та текстом, щоб спрощувати кодування, потрібне для написання глибинно-нейромережного коду. На додачу до стандартних нейронних мереж, Keras містить підтримку згорткових та рекурентних нейронних мереж. Вона підтримує інші поширені службові шари, такі як виключення та агрегування. Keras дає своїм користувачам можливість виробляти продукти на основі глибинних моделей для смартфонів (iOS та Android), вебсайтів та віртуальної машини Java.

NumPy — open-source модуль для Python, який надає загальні математичні і числові операції у вигляді пре-скомпільовані, швидких функцій. Вони об'єднуються в високорівневі пакети. Вони забезпечують функціонал, який можна порівняти з функціоналом MatLab. NumPy (Numeric Python) надає базові методи для маніпуляції з великими масивами і матрицями. SciPy (Scientific Python) розширює функціонал numpy величезною колекцією корисних алгоритмів, таких як мінімізація, перетворення Фур'є, регресія, і інші прикладні математичні техніки [16].

Pandas — фреймворк, створений для мови програмування Python для роботи з даними та їх аналізу. Він дозволяє працювати з файлами як з структурами. За допомогою pandas можна легко та зручно обробляти великі файли. Також перевагою серед інших інструментів є багато можливостей для аналізу файлових баз даних та часових рядів [16].

В ході тестування програмного продукту були визначені наступні вимоги щодо ПК:

1. Операційна система Windows x64, Ubuntu та MacOS.
2. 8 гб оперативної пам’яті.
3. Intel Core i7-5960X Processor або вище.
4. Keras, numpy, pandas.
5. Встановлене середовище для мови програмування Python, бажано Jupyter.

Приведемо опис частин програми.

1. Модуль завантаження даних - даний модуль являє собою набір функцій, призначених для завантаження вхідних даних та містить класи і функції, що полегшують доступ до конкретних значень та параметрів вибірки. Модуль pandas використовується для зберігання даних у датасеті з .csv файла отриманої інформації.
2. Модуль роботи із архітектурами - в даному модулі реалізовані алгоритми та архітектури методів машинного навчання, а саме – EfficientNet, ResNet, VGG19.

3.4 Порівняння результатів

Використовуючи Keras була побудована (рис. 3.9) та натренована EfficientNet модель (Рис. 3.10).

```
def modelEfficientNetB3():

    model = models.Sequential()
    model.add(EfficientNetB3(include_top = False, weights = "./efficientnetb3_notop.h5",input_shape=(IMG_SIZE,IMG_SIZE, 3)))
    model.add(layers.GlobalAveragePooling2D())
    model.add(layers.Dense(256, activation = 'relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(5, activation = "softmax"))

    return model
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
efficientnetb3 (Functional)  (None, 10, 10, 1536)      10783535
_________________________________________________________________
global_average_pooling2d (Gl (None, 1536)              0
_________________________________________________________________
dense (Dense)                (None, 256)               393472
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 1285
=================================================================
```

```
model.compile(optimizer = "adam",
              loss = CategoricalCrossentropy(label_smoothing=0.3,reduction="auto",name="categorical_crossentropy"),
              metrics = ["accuracy"])
```

Рисунок 3.9 – Побудована EfficientNet модель

```
Epoch 1/30
535/535 [==============================] - ETA: 0s - loss: 1.2023 - accuracy: 0.7545
Epoch 00001: val_loss improved from inf to 1.22667, saving model to ./firstTry.h5
535/535 [==============================] - 818s 2s/step - loss: 1.2023 - accuracy: 0.7545 - val_loss: 1.2267 - val_accuracy: 0.703
4
Epoch 2/30
535/535 [==============================] - ETA: 0s - loss: 1.1279 - accuracy: 0.8128
Epoch 00002: val_loss improved from 1.22667 to 1.16089, saving model to ./firstTry.h5
535/535 [==============================] - 724s 1s/step - loss: 1.1279 - accuracy: 0.8128 - val_loss: 1.1609 - val_accuracy: 0.788
0
Epoch 3/30
535/535 [==============================] - ETA: 0s - loss: 1.1089 - accuracy: 0.8269
Epoch 00003: val_loss improved from 1.16089 to 1.12905, saving model to ./firstTry.h5
535/535 [==============================] - 727s 1s/step - loss: 1.1089 - accuracy: 0.8269 - val_loss: 1.1291 - val_accuracy: 0.805
8
Epoch 4/30
535/535 [==============================] - ETA: 0s - loss: 1.0951 - accuracy: 0.8391
Epoch 00004: val_loss did not improve from 1.12905
535/535 [==============================] - 727s 1s/step - loss: 1.0951 - accuracy: 0.8391 - val_loss: 1.1522 - val_accuracy: 0.779
6
Epoch 5/30
535/535 [==============================] - ETA: 0s - loss: 1.0905 - accuracy: 0.8413
Epoch 00005: val_loss improved from 1.12905 to 1.08284, saving model to ./firstTry.h5
535/535 [==============================] - 729s 1s/step - loss: 1.0905 - accuracy: 0.8413 - val_loss: 1.0828 - val_accuracy: 0.836
4
Epoch 6/30
535/535 [==============================] - ETA: 0s - loss: 1.0832 - accuracy: 0.8483
Epoch 00006: val_loss did not improve from 1.08284
535/535 [==============================] - 732s 1s/step - loss: 1.0832 - accuracy: 0.8483 - val_loss: 1.0829 - val_accuracy: 0.835
7
Epoch 7/30
535/535 [==============================] - ETA: 0s - loss: 1.0726 - accuracy: 0.8537
Epoch 00007: val_loss improved from 1.08284 to 1.06543, saving model to ./firstTry.h5
535/535 [==============================] - 736s 1s/step - loss: 1.0726 - accuracy: 0.8537 - val_loss: 1.0654 - val_accuracy: 0.853
7
```

```
Epoch 8/30
535/535 [==============================] - ETA: 0s - loss: 1.0701 - accuracy: 0.8549
Epoch 00008: val_loss improved from 1.06543 to 1.05586, saving model to ./firstTry.h5
535/535 [==============================] - 735s 1s/step - loss: 1.0701 - accuracy: 0.8549 - val_loss: 1.0559 - val_accuracy: 0.861
2
Epoch 9/30
535/535 [==============================] - ETA: 0s - loss: 1.0709 - accuracy: 0.8558
Epoch 00009: val_loss did not improve from 1.05586
535/535 [==============================] - 737s 1s/step - loss: 1.0709 - accuracy: 0.8558 - val_loss: 1.0580 - val_accuracy: 0.861
9
Epoch 10/30
535/535 [==============================] - ETA: 0s - loss: 1.0655 - accuracy: 0.8605
Epoch 00010: val_loss did not improve from 1.05586

Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
535/535 [==============================] - 741s 1s/step - loss: 1.0655 - accuracy: 0.8605 - val_loss: 1.0902 - val_accuracy: 0.830
6
Epoch 11/30
535/535 [==============================] - ETA: 0s - loss: 1.0412 - accuracy: 0.8803
Epoch 00011: val_loss improved from 1.05586 to 1.03640, saving model to ./firstTry.h5
535/535 [==============================] - 745s 1s/step - loss: 1.0412 - accuracy: 0.8803 - val_loss: 1.0364 - val_accuracy: 0.875
7
Epoch 12/30
535/535 [==============================] - ETA: 0s - loss: 1.0306 - accuracy: 0.8863
Epoch 00012: val_loss did not improve from 1.03640
535/535 [==============================] - 737s 1s/step - loss: 1.0306 - accuracy: 0.8863 - val_loss: 1.0388 - val_accuracy: 0.872
9
Epoch 13/30
535/535 [==============================] - ETA: 0s - loss: 1.0275 - accuracy: 0.8906
Epoch 00013: val_loss improved from 1.03640 to 1.03488, saving model to ./firstTry.h5
535/535 [==============================] - 746s 1s/step - loss: 1.0275 - accuracy: 0.8906 - val_loss: 1.0349 - val_accuracy: 0.874
3
```

```
Epoch 14/30
535/535 [==============================] - ETA: 0s - loss: 1.0225 - accuracy: 0.8937
Epoch 00014: val_loss did not improve from 1.03488
535/535 [==============================] - 746s 1s/step - loss: 1.0225 - accuracy: 0.8937 - val_loss: 1.0350 - val_accuracy: 0.876
1
Epoch 15/30
535/535 [==============================] - ETA: 0s - loss: 1.0179 - accuracy: 0.8984
Epoch 00015: val_loss improved from 1.03488 to 1.03477, saving model to ./firstTry.h5
535/535 [==============================] - 759s 1s/step - loss: 1.0179 - accuracy: 0.8984 - val_loss: 1.0348 - val_accuracy: 0.873
6
```

Рисунок 3.10 – Навчання мережі EfficientNet

Видно, що після 15 епохи функція втрат не змінювалась, тому навчання було призупинено. Проаналізуємо значення Accuracy на тренувальній та валідаційній вибірках.

Як було очікувано, після довгого тренування Accuracy для набору даних train та validation перевищила 0.85, що є дуже позитивним результатом (Рис. 3.11). Loss для двох виборок також показав гарні результати (Рис. 3.12).

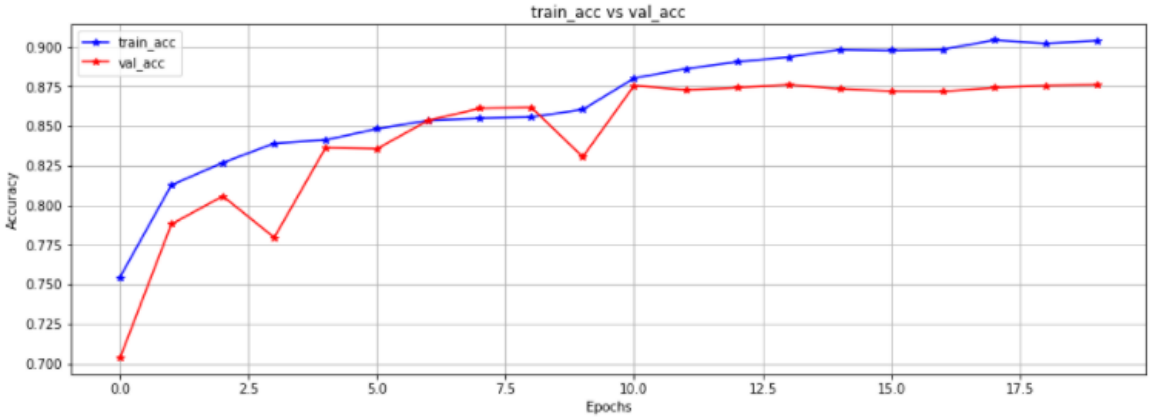


Рисунок 3.11 – Accuracy для train та validation

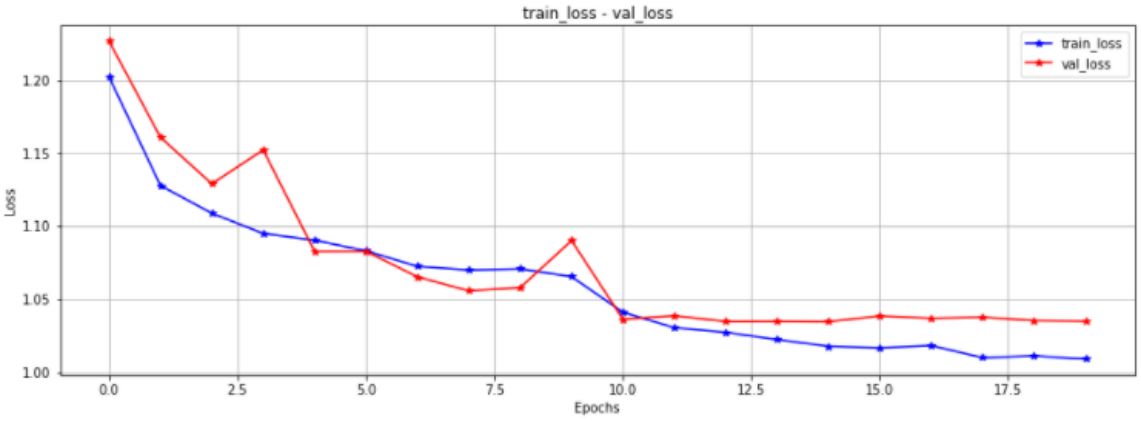


Рисунок 3.12 – Loss для train та validation

Тепер розглянемо роботу мережі VGG19 (рисунок 3.13) та як вона впоралась із нашими даними.

```
base_model = keras.applications.vgg19.VGG19(
    weights="imagenet",  # Load weights pre-trained on ImageNet.
    input_shape=(256, 256, 3),
    include_top=False,
)
base_model.trainable = False # freeze base_model

input_shape = (256, 256, 3)
inputs = keras.Input(shape=input_shape)
scale_layer = keras.layers.Rescaling(scale=1./127.5, offset=-1) # [-1, 1]
x = scale_layer(inputs)
#x=inputs/255

x = base_model(x, training=False)
x = keras.layers.GlobalAveragePooling2D()(x)
x = keras.layers.Dropout(0.5)(x)  # Regularize with dropout
outputs = keras.layers.Dense(5, activation=classification_type)(x)
model = keras.Model(inputs, outputs)
model.summary()
```

```
model.compile(
loss='categorical_crossentropy',
optimizer=optimizers.RMSprop(lr=1e-4),
metrics=['acc'])

#BATCH_SIZE = 16
epochs = 10
#creuser ça
filepath = "VGG19_TPUmodel.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1,
                             save_best_only=True, mode='max')
reduce_lr = ReduceLROnPlateau(monitor='val_acc', factor=0.5, patience=2,
                                verbose=1, mode='max', min_lr=0.00001)
callbacks_list = [checkpoint, reduce_lr]

history = model.fit(
    train_gen,
    steps_per_epoch=train_steps,
    validation_data=val_gen,
    validation_steps=val_steps,
    epochs = epochs,
    callbacks=callbacks_list)
```

Рисунок 3.13 – Побудова мережі VGG19

Розглянемо як ця мережа навчається на рисунку 3.14.

```
94/94 [==============================] - 97s 911ms/step - loss: 1.4063 - acc: 0.4308 - val_loss: 1.2390 - val_acc: 0.5966

Epoch 00001: val_acc improved from -inf to 0.59657, saving model to VGG19_TPUmodel.h5
Epoch 2/10
94/94 [==============================] - 80s 853ms/step - loss: 1.2678 - acc: 0.5979 - val_loss: 1.2200 - val_acc: 0.5966

Epoch 00002: val_acc did not improve from 0.59657
Epoch 3/10
94/94 [==============================] - 81s 865ms/step - loss: 1.2439 - acc: 0.6159 - val_loss: 1.2222 - val_acc: 0.5966

Epoch 00003: val_acc did not improve from 0.59657

Epoch 00003: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
Epoch 4/10
94/94 [==============================] - 81s 864ms/step - loss: 1.2400 - acc: 0.6184 - val_loss: 1.2230 - val_acc: 0.5966

Epoch 00004: val_acc did not improve from 0.59657
Epoch 5/10
94/94 [==============================] - 81s 864ms/step - loss: 1.2369 - acc: 0.6209 - val_loss: 1.2225 - val_acc: 0.5966

Epoch 00005: val_acc did not improve from 0.59657

Epoch 00005: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-05.
Epoch 6/10
94/94 [==============================] - 79s 845ms/step - loss: 1.2257 - acc: 0.6211 - val_loss: 1.2224 - val_acc: 0.5966

Epoch 00006: val_acc did not improve from 0.59657
Epoch 7/10
94/94 [==============================] - 81s 859ms/step - loss: 1.2316 - acc: 0.6213 - val_loss: 1.2227 - val_acc: 0.5966

Epoch 00007: val_acc did not improve from 0.59657

Epoch 00007: ReduceLROnPlateau reducing learning rate to 1.249999968422344e-05.
Epoch 8/10
94/94 [==============================] - 81s 865ms/step - loss: 1.2282 - acc: 0.6214 - val_loss: 1.2225 - val_acc: 0.5966

Epoch 00008: val_acc did not improve from 0.59657
Epoch 9/10
94/94 [==============================] - 80s 852ms/step - loss: 1.2260 - acc: 0.6214 - val_loss: 1.2227 - val_acc: 0.5966

Epoch 00009: val_acc did not improve from 0.59657

Epoch 00009: ReduceLROnPlateau reducing learning rate to 1e-05.
Epoch 10/10
94/94 [==============================] - 81s 860ms/step - loss: 1.2368 - acc: 0.6216 - val_loss: 1.2226 - val_acc: 0.5966

Epoch 00010: val_acc did not improve from 0.59657
```

Рисунок 3.14 – Тренування мережі VGG19

Бачимо, що хоча ця архітектура і покращує результат, він змінюється не довго і все одно залишається лише задовільним та дає точність у 0.6. Проте прослідкуємо, як себе веде Accuracy та Loss на двух вибірках (рис. 3.15).

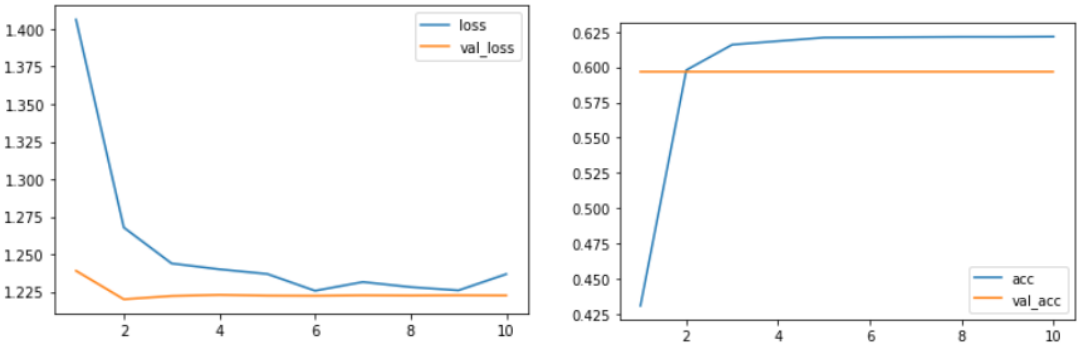


Рисунок 3.15 – Accuracy та Loss для двух виборок

Використовуючи вже існуючу архітектуру була використана (рис. 3.9) та натренована ResNet модель (Рис. 3.16).

```
EVAL: [133/134] Data 0.000 (0.203) Elapsed 0m 40s (remain 0m 0s) Loss: 0.2571(0.3658)
Epoch: [10][0/534] Data 1.307 (1.307) Elapsed 0m 1s (remain 15m 31s) Loss: 0.2900(0.2900) Gr
ad: 4.4743
Epoch: [10][100/534] Data 0.003 (0.014) Elapsed 0m 42s (remain 3m 3s) Loss: 0.3071(0.3160) G
rad: 4.9580
Epoch: [10][200/534] Data 0.000 (0.008) Elapsed 1m 23s (remain 2m 18s) Loss: 0.3406(0.3075)
Grad: 4.9168
Epoch: [10][300/534] Data 0.000 (0.005) Elapsed 2m 3s (remain 1m 35s) Loss: 0.1916(0.3019) G
rad: 3.8441
Epoch: [10][400/534] Data 0.000 (0.004) Elapsed 2m 43s (remain 0m 54s) Loss: 0.2736(0.3035)
Grad: 5.6447
Epoch: [10][500/534] Data 0.002 (0.004) Elapsed 3m 24s (remain 0m 13s) Loss: 0.2669(0.3003)
Grad: 4.6641
Epoch: [10][533/534] Data 0.000 (0.003) Elapsed 3m 37s (remain 0m 0s) Loss: 0.4307(0.3016) G
rad: 6.6076
EVAL: [0/134] Data 1.349 (1.349) Elapsed 0m 1s (remain 3m 12s) Loss: 0.3359(0.3359)
EVAL: [100/134] Data 0.667 (0.216) Elapsed 0m 31s (remain 0m 10s) Loss: 0.3647(0.3643)


Epoch 10 - avg_train_loss: 0.3016  avg_val_loss: 0.3666  time: 259s
Epoch 10 - Accuracy: 0.8719326945548025
```

Рисунок 3.16 – Навчання мережі ResNet

Маючи усі дані, порівняємо їх у таблиці 3.2, для кращого сприйняття.

Таблиця 3.2 – Порівняння моделей

Модель	Accuracy	Loss by train	Accuracy by validation	Loss by validation
EfficientNet	0.9040	1.0092	0.876	1.0351
VGG19	0.6216	1.2368	0.5966	1.2226
ResNet	0.873	0.3	0.8719	0.366

Існуючі моделі є найкращими для рішення цієї задачі. Ми бачимо що перша модель дає вищу точність, проте функція витрат також показує високий результат. Можна зробити висновок, що ResNet є більш оптимальною для класифікації цих зображень.

3.5 Висновки до розділу 3

В цьому розділі було проведено практичне моделювання.

Було поведено обробку даних, нормалізацію датасету та зображень. Потім було навчено 3 моделі згорткової нейронної мережі для задачі класифікації хвороби листя маніоки. Результати моделювання оцінювались метриками Accuracy, Loss. Графіки для результатів кожної моделі у вигляді кривої помилок.

В результаті найкращим методом стала модель ResNet та показала гарний результат. Точність була більша за 0.85. Алгоритми успішно класифікували різні хвороби у правильні класи. Навчання відбувалося на даних train, та перевірена і протестована на validation, test. Було використано Python 3.9 та бібліотеки Keras, Numpy та Pandas, оскільки ці інструментарії найкраще підходять для виконання поставленої задачі, а також для проектів машинного навчання та штучного інтелекту.

Отримана точність є достатньо високою. Збільшити її можливо в ході подальших досліджень шляхом об’єднання різних класифікаторів, додатковою обробкою початкових даних тощо.

4 РОЗРОБКА СТАРТАП-ПРОЕКТУ

Стартап-проект — є комерційним проектом, який знаходиться в стані розробки, або нещодавно вийшов на ринок. Характерною особливістю стартапу, що відрізняє його від малого бізнесу, є оригінальність та інновації, він не може бути копією вже реалізованих ідей. Також, зазвичай, розробкою і розвитком продукту, послуги, ідеї, яку пропонує тільки що заснована молода фірма завжди займається команда однодумців.

У стартапу не дуже міцне положення на ринку і не завжди вистачає капіталу на його запуск, що є слабкою стороною. В Європі подібні проекти створюються і втілюються в життя студентами. Часом такі фірми іменуються «гаражними». Для того, щоб залучити інвесторів і свою аудиторію, ваш стартап повинен бути успішним і затребуваним. Якщо ваші інвестори готові і хочуть робити вкладення в стартап, значить, ви на вірному шляху. Ваші інноваційні ідеї дадуть непоганий результат. Головним завданням стартапу на початковому етапі його існування – є перетворення ідеї проекту у працюючу бізнес-модель: знайти згуртовану команда розробників і помічників, а також достатнє фінансування аж до тієї пори, коли проект не стане самоокупним і прибутковим

Розроблення та виведення стартап-проекту на ринок передбачає здійснення низки кроків, в межах яких визначають ринкові перспективи проекту, графік та принципи організації виробництва, фінансовий аналіз та аналіз ризиків і заходи з просування пропозиції для інвесторів. Далі наведено маркетинговий аналіз стартап проекту.

В межах цього етапу:

1. Розробляється опис самої ідеї проекту та визначаються загальні напрями використання потенційного товару чи послуги, а також їх відмінність від конкурентів.

2. Аналізуються ринкові можливості щодо його реалізації.
3. На базі аналізу ринкового середовища розробляється стратегія ринкового впровадження потенційного товару в межах проекту.

4.1. Опис ідеї проекту

В межах підпункту було проаналізовано і подано у вигляді таблиць:

1) зміст ідеї (що пропонується),
2) можливі напрямки застосування,
3) основні вигоди, що може отримати користувач товару (за кожним напрямком застосування),
4) чим відрізняється від існуючих аналогів та замінників.

Перші три пункти подані у вигляді таблиці (таблиця 4.1) і дають цілісне уявлення про зміст ідеї та можливі базові потенційні ринки, в межах яких потрібно шукати групи потенційних клієнтів.

Таблиця 4.1 - Опис ідеї стартап-проекту

Зміст ідеї	**Напрямки застосування**	**Вигоди для користувача**
Розв’язати проблему розповсюдження хвороби коренеплідної рослини – маніоки	1. Попередження розповсюдження хвороби	Зменшення кількості непридатних до споживання плодів

Продовження таблиці 4.1

	2. Виявлення та класифікація хвороби	Швидке визначення хвороби для оперативних подальших дій

Аналіз потенційних техніко-економічних переваг ідеї (чим відрізняється від існуючих аналогів та замінників) порівняно із пропозиціями конкурентів передбачає:

1) визначення переліку техніко-економічних властивостей та характеристик ідеї,
2) визначення попереднього кола конкурентів (проектів-конкурентів) або товарів-замінників чи товарів-аналогів, що вже існують на ринку, та проводиться збір інформації щодо значень техніко-економічних показників для ідеї власного проекту та проектів-конкурентів відповідно до визначеного вище переліку,
3) проводиться порівняльний аналіз показників: для власної ідеї визначаються показники, що мають а) гірші значення (W, слабкі); б) аналогічні (N, нейтральні) значення; в) кращі значення (S, сильні) (табл. 4.2).

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

Технікоекономічні характеристики ідеї	**(потенційні) товари/концепції конкурентів**			**W**	**N**	**S**
	Мій проект	**Agrobase**	**Plantix**			
Форма виконання	Надавання послуг	Надавання послуг	Надавання послуг		+	
Собівартість	Низька	Висока	Висока			+
Функціонал	Вузький	Широкий	Широкий	+		

Визначений перелік слабких, сильних та нейтральних характеристик та властивостей ідеї потенційного товару є підґрунтям для формування його конкурентоспроможності.

4.2 Технологічний аудит ідеї проекту

В межах даного підрозділу було проведено аудит технології, за допомогою якої можна реалізувати ідею проекту (технології створення товару). Визначення технологічної здійсненності ідеї проекту передбачає аналіз таких складових (таблиця 4.3):

1. За якою технологією буде виготовлено товар згідно ідеї проекту.
2. Чи існують такі технології, чи їх потрібно розробити/доробити.
3. Чи доступні такі технології авторам проекту.

Таблиця 4.3 – Технологічна здійсненність ідеї проекту

Ідея проекту	**Технології її реалізації**	**Наявність технологій**	**Доступність технологій**
Створення системи, що визначає хворобу по зображенню	Використання мови програмування Python (середа Jupyter)	Наявна	Доступна
	Використання мови програмування C++	Відсутня	Недоступна
	Використання мови програмування C#	Відсутня	Недоступна
	Обрана технологія реалізації ідеї проекту: мова програмування Python (середа Jupyter)		

За результатами аналізу таблиці зроблено висновок щодо можливості технологічної реалізації проекту. Технологічним шляхом реалізації проекту було обрано такі технології, як Python 3.9 та Jupyter через їх доступність та безкоштовність.

Визначення ринкових можливостей, які можна використати під час 77 ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

Спочатку було проведено аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку (таблиця 4.4).

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проекту

Показники стану ринку (найменування)	**Характеристика**
Кількість головних гравців, од	45
Загальний обсяг продаж, грн/ум.од	90000
Динаміка ринку (якісна оцінка)	Зростає
Наявність обмежень для входу (вказати характер обмежень)	Немає
Специфічні вимоги до стандартизації та сертифікації	Немає
Середня норма рентабельності в галузі (або по ринку), %	12%

Середню норму рентабельності в галузі було порівняно із банківським відсотком на вкладення. Останній є меншим, тому є сенс вкладати гроші саме у цей проект.

За результатами аналізу таблиці 4.4 було зроблено висновок, що ринок є привабливим для входження.

Надалі були визначені потенційні групи клієнтів, їх характеристики та сформовано орієнтовний перелік вимог до товара для кожної групи (табл. 4.5).

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проекту

Потреба, що формує ринок	**Цільова аудиторія (цільові сегменти ринку)**	**Відмінності у поведінці різних потенційних цільових груп клієнтів**	**Вимоги споживачів до товару**
Точне та швидке виявлення хвороби рослини	Фермери, прості користувачі	Потреба у простому обладнанні	Простота використання, висока точність
Швидкий та якісний аналіз хвороби	Агро-компанії, фермерські компанії	Велика кількість даних	Швидкість створення, низька ціна

Після визначення потенційних груп клієнтів було проведено аналіз ринкового середовища: складено таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (табл. 4.6, 4.7).

Таблиця 4.6 - Фактори загроз

Фактор	**Зміст загрози**	**Можлива реакція компанії**
Конкуренція	Вихід на ринок продуктів з більшим функціоналом	Передбачити розвиток ПП у функціональності, мати додаткові переваги. Передбачити можливість вдосконалення власного продукту. Проаналізувати інші цільові сегменти.

Продовження таблиці 4.6

Збільшення потреб користувачів	Користувачам необхідний сервіс з більшим/новим функціоналом.	Розроблення гнучкої архітектури програмного забезпечення для легшого впровадження нового функціоналу

Таблиця 4.7 - Фактори можливостей

Фактор	**Зміст можливості**	**Можлива реакція компанії**
Гнучкі ціни	Зменшення ціни товару за для збільшення попиту	Введення власних гнучких цін
Поява швидших або точніших методів розпізнавання	З’являться нові методи, що будуть швидше, та більш точно	Покращити ПП додаванням нового функціоналу, розширення можливостей

Надалі було проведено аналіз пропозиції: визначили загальні риси конкуренції на ринку (таблиця 4.8).

Таблиця 4.8 - Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	**В чому проявляється дана характеристика**	**Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)**
Чиста конкуренція	Багато додатків та програм, що можуть вирішити таку задачу	Розробити впізнаваний продукт, якість, що вирізнятиме нас від конкурентів
Міжнародний рівень конкурентної боротьби	Компанії-конкуренти з інших країн	Розширення аудиторії, розширення списку мов, які підтримуються системою
Міжгалузева конкуренція	Розпізнавання об’єктів застосовується не лише агроіндустрії	Можливість вести бізнес в ширшому діапазоні галузей
Товарно-родова конкуренція	Створення товарів однієї категорії	Ведення активної рекламної кампанії
Нецінова конкурентна перевага	Вдосконалення технології створення ПП, щоб собівартість була нижчою	Удосконалення моделі. Використання більш дешевих технологій для розробки, ніж використовують конкуренти, але тільки якщо ці технології відповідають необхідним вимогам якості.

Продовження таблиці 4.8

Марочна	Система призначена для спільної цільової аудиторії	Реклама, участь у конференціях, семінарах.

Проведемо аналіз конкуренції у галузі за моделлю М. Портера (табл. 4.9).

Таблиця 4.9 - Аналіз конкуренції в галузі за М. Портером

	Прямі конкуренти в галузі	**Потенційні конкуренти**	**Постача-льники**	**Клієнти**	**Товаризамінники**
Складові аналізу	Agrobase Plantix Pestoz.	Наявність вже існуючих рішень	-	Фермери, агро-компанії	Наявність більш широкого функціоналу, зручнішого інтерфейсую
Висновки	Досить інтенсивна конкурентна боротьба з іншими гравцями	Є можливості виходу на ринок, але є і конкуренти.	-	Клієнти диктують умови роботи на ринку: висока точність, велика база хвороб	Необхідно випускати ПП не гірше, ніж у конкурентів та розширяти функціонал.

За результатами аналізу було зроблено висновок про можливість роботи на ринку з огляду на конкурентну ситуацію. Також було зроблено висновок щодо характеристик, які повинен мати проект, щоб бути конкурентоспроможним на ринку.

Цей висновок був врахований при формулюванні переліку факторів конкурентоспроможності у наступному пункті. На основі аналізу конкуренції, проведеного в таблиці, а також із урахуванням характеристик ідеї проекту (табл. 4.2), вимог споживачів до товару (табл. 4.5) та факторів маркетингового середовища (табл. 4.6, 4.7) визначається та обґрунтовується перелік факторів конкурентоспроможності. Аналіз оформлено у (табл. 4.10).

Таблиця 4.10 - Обґрунтування факторів конкурентоспроможності

Фактор конкурентоспроможності	**Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)**
Низька собівартість	Використання лише звичайної камери, суттєво знижують витрати на комплектуючі
Якість	Висока якість прогнозу, велика кількість допоміжних статистичних даних
Зручність та простота роботи з програмою	Дозволяє користувачу легко працювати з програмою, маючи при цьому тільки свій телефон

За визначеними факторами конкурентоспроможності (табл. 4.10) проведено аналіз сильних та слабких сторін стартап-проекту (табл. 4.11).

Таблиця 4.11 - Порівняльний аналіз сильних та слабких сторін проекту

Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні						
		–3	–2	–1	0	+1	+2	+3
Низька собівартість	15					*		
Якість	10			*				
Зручність роботи з програмою	5					*		

Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) (табл. 4.12) на основі виділених ринкових загроз та можливостей, та сильних і слабких сторін (таблиця 4.11).

Перелік ринкових загроз та ринкових можливостей було складено на основі аналізу факторів загроз та факторів можливостей маркетингового середовища. Ринкові загрози та ринкові можливості є наслідками (прогнозованими результатами) впливу факторів, і, на відміну від них, ще не є реалізованими на ринку та мають певну ймовірність здійснення. Наприклад: зниження доходів потенційних споживачів – фактор загрози, на основі якого можна зробити прогноз щодо посилення значущості цінового фактору при виборі товару та відповідно, – цінової конкуренції (а це вже – ринкова загроза).

Таблиця 4.12 - SWOT-аналіз стартап-проекту

Сильні сторони: Якість, простота використання, висока швидкодія, низька собівартість	Слабкі сторони: Дуже насичений ринок, мала кількість функціоналу
Можливості: Насичення ринку новим підходом до прогнозування; різноманітна клієнтура, вдосконалення системи	Загрози: Значна конкуренція

На основі SWOT-аналізу було розроблено альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок. Визначені альтернативи були проаналізовані з точки зору строків та ймовірності отримання ресурсів (таблиця 4.13).

Таблиця 4.13 - Альтернативи ринкового впровадження стартап-проекту

Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
Вихід на ринок із «сирим» продуктом, проблеми із неточністю прогнозу	20%	2 місяці

Продовження таблиці 4.13

Поступовий вихід із готовим, продуманим проектом Висока точність та якісне прогнозування	70%	5 місяців
Партнерство для об’єднання продукції	10%	2 місяці

Після аналізу було обрано альтернативу №2.

4.3 Аналіз ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: було проведено опис цільових груп потенційних споживачів (таблиця 4.14).

Таблиця 4.14 - Вибір цільових груп потенційних споживачів

Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
Фермери	Висока	Високий	Сильна	Просто

Продовження таблиці 4.14

Поодинокі користувачі	Середня	Низька	Сильна	Складно
Великі компанії	Середня	Середній	Сильна	Складно
Маленькі компанії	Середня	Низький	Слабка	Середня

За результатами аналізу потенційних груп споживачів було обрано цільові групу, для яких буде запропоновано даний товар, та визначено стратегію охоплення ринку - стратегію диференційованого маркетингу (компанія працює з декількома сегментами).

Для роботи в обраних сегментах ринку сформовано базову стратегію розвитку (таблиця 4.15).

Таблиця 4.15 - Визначення базової стратегії розвитку

Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
Постійне оновлення і покращення продукту	Ринкове позиціонування на індивідуальних користувачів	Швидкодія, якість продукту, низька собівартість	Концентрований маркетинг та покращення функціональності

Наступним кроком обрано стратегію конкурентної поведінки (таблиця 4.16).

Таблиця 4.16 - Визначення базової стратегії конкурентної поведінки

Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента і які?	Стратегія конкурентної поведінки*
Ні.	Так	Частково	Зайняття конкурентної ніші

На основі вимог споживачів з обраних сегментів до постачальника (стартап-компанії) та до продукту (табл. 4.5), а також в залежності від обраної базової стратегії розвитку (табл. 4.15) та стратегії конкурентної поведінки (таблиця 4.16) розроблено стратегію позиціонування (таблиця 4.17), що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 4.17 - Визначення стратегії позиціонування

Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
Легкість розуміння, зручний інтерфейс, надійний, швидкий, точний та достовірний ПП для створення палітри кольорів.	Стратегія диференціації	Позиція на основі порівняння фірми з товарами конкурентів; Відмінні особливості споживача	Економія часу; Зручність застосування; Практичність та точність результату

Результатом виконання підрозділу стала узгоджена система рішень щодо ринкової поведінки стартап-компанії, яка визначає напрями роботи стартап- компанії на ринку.

4.4 Розроблення маркетингової програми стартап-проекту

Сформовано маркетингову концепцію товару, який отримає споживач. Для цього підсумовано результати попереднього аналізу

конкурентоспроможності товару (таблиця 4.18). Концепція товару - письмовий опис фізичних та інших характеристик товару, які сприймаються споживачем, і набору вигод, які він обіцяє певній групі споживачів.

Таблиця 4.18 – Визначення ключових переваг концепції потенційного товару

Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
Швидкість отримання результату	Система надає можливість розпізнавання в реальному часі	Необхідно додатково підвищити швидкість розпізнавання для отримання переваги
Зручність застосування	Не потрібно мати спеціального обладнання	Для даного ПП потрібно мати лише свій власний телефон із камерою
Практичність та точність результату	Користувач отримує точні (з малою похибкою розбіжності) результати.	Розробка коротко- та довгострокових прогнозів, використання різних методів для покращення точності прогнозу

Розроблено трирівневу маркетингову модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання (таблиця 4.19).

1-й рівень: При формуванні задуму товару вирішується питання щодо того, засобом вирішення якої потреби і / або проблеми буде даний товар, яка

його основна вигода. Дане питання безпосередньо пов'язаний з формуванням технічного завдання в процесі розробки конструкторської документації на виріб.

2-й рівень: Цей рівень являє рішення того, як буде реалізований товар в реальному/ включає в себе якість, властивості, дизайн, упаковку, ціну.

3-й рівень: Товар з підкріпленням (супроводом) - додаткові послуги та переваги для споживача, що створюються на основі товару за задумом і товару в реальному виконанні (гарантії якості , доставка, умови оплати та ін).

Таблиця 4.19 - Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>I. Товар за задумом</td><td colspan="3">Класифікація хвороби зображення маніоки за допомогою інтелектуальних систем</td></tr>
<tr><td rowspan="5">II. Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх/Тл/Е/Ор</td></tr>
<tr><td>1. Індивідуальний підхід.
2. Низька ціна.
3. Простота у використанні.</td><td>1.Нм
2.Нм
3.Нм</td><td>1.Технологічна
2.Економічна
3.Технологічна</td></tr>
<tr><td colspan="3">Якість: тестування сторонніми фірмами</td></tr>
<tr><td colspan="3">Пакування: відсутнє</td></tr>
<tr><td colspan="3">Марка: DZH</td></tr>
<tr><td rowspan="2">III. Товар із підкріпленням</td><td colspan="3">До продажу: відсутнє</td></tr>
<tr><td colspan="3">Після продажу: персональна підтримка в обслуговуванні за додаткову платню.</td></tr>
<tr><td colspan="4">За рахунок чого потенційний товар буде захищено від копіювання: ліцензія.</td></tr>
</table>

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар (остаточне визначення ціни відбувається під час фінансово-економічного аналізу проекту), яке передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів (таблиця 4.20). Аналіз проведено експертним методом.

Таблиця 4.20 - Визначення меж встановлення ціни

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1400$	2100$	У всіх трьох груп високий рівень доходів	500-1000$

Наступним кроком є визначення оптимальної системи збуту, в межах якого було прийняте рішення (таблиця 4.21).

Таблиця 4.21 - Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Канал нульового рівня	Продаж	0(напряму)	Власна

Останньою складової маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (таблиця 4.22).

Таблиця 4.22 - Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
Встановлення програми на телефон	Інтернет	Низька ціна, простота використання, універсальність	Показати переваги рішення над конкурентами, виділити ключові особливості	Створення сайту продукту, розповсюдження інформації про продукт на спеціалізованих ресурсах.

Було визначено, що придбання продукту буде проводитись через мережу Інтернет. Розповсюдження інформації про продукт буде проводитись виключно через Інтернет, адже аудиторія даного продукту активно користується всесвітньою мережею.

Результатом підрозділу стала ринкова (маркетингова) програма, що включає в себе концепції товару, збуту, просування та попередній аналіз можливостей ціноутворення, спирається на цінності та потреби потенційних клієнтів, конкурентні переваги ідеї, стан та динаміку

ринкового середовища, в межах якого впроваджено проект, та відповідну обрану альтернативу ринкової поведінки.

4.5 Висновки до розділу 4

В даному розділі проведено підготовчий аналіз для впровадження розробленої системи в якості стартап проекту. Досліджено аналогічні конкурентні системи, встановлено сильні та слабкі сторони системи в порівнянні з ними. Також було досліджено можливі шляхи розповсюдження продукту та його ймовірну аудиторію, рівень доходів та ймовірну ціну продукту, що розробляється.

Було проведено аналіз потенційних ризиків і можливостей, а також розраховані основні фінансово-економічні показники проекту. Отримані результати кажуть про те, що реалізація проекту є доцільною. Було визначено сильні сторони проекту: зручність у використанні, ціна, якість. Серед слабких варто виділити невелику кількість функціоналу. Варто відмітити можливість реклами продукту на спеціалізованих ресурсах із зазначенням сильних сторін проекту.

ВИСНОВКИ ДО РОБОТИ

У ході виконання даної магістерської дисертації, були досліджені різноманітні методи визначення хвороби сільськогосподарської рослини, а саме маніоки. Зупинившись на алгоритмі, що використовує нейронні мережі, було досліджено головні аспекти даного способу та проаналізовані найкращі підходи до навчання. Крім того, була проведена порівняльна характеристика прийомів побудови моделі нейронних мереж.

В роботі було запропоновано модель глибокого навчання для вияву та класифікації хвороб маніоки, яка одночасно покладається на особливості текстури, структури та форми. Експериментальні результати вказують на те, що модель добре працює зі складним набором даних. Було проведено експерименти, в ході яких виявлено архітектуру згорткової нейронної мережі для класифікації та сегментації зображень листя маніоки.

Отже, запропонована система може відігравати ключове значення при класифікації та сегментації зображень листя. Для подальшого підвищення ефективності моделі можна було виділено наступні напрямки подальших досліджень:

1. Зробити налаштування гіперпараметрів та кращу техніку попередньої обробки даних.
2. Використати інші оптимізатори та перевірити їх вплив на результати.
3. Проводити дослідження щодо більш збалансованого набору даних.
4. Слід дослідити можливість використання більшої кількості зображень та класів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Ayu H. R. Deep learning for detection cassava leaf disease. IOP Publishing. 2021. No.1751. P. 2–3.
2. Cassava Leaf Disease Classification Using Convolutional Neural Networks. 2021. URL: https://medium.com/mlearning-ai/cassava-leaf-disease-classification-using-convolutional-neural-networks-48fdc32cec1d.
3. Ramcharan A. Deep Learning for Image-Based Cassava Disease Detection. 2017.URL:https://www.frontiersin.org/articles/10.3389/fpls.2017.01852/full
4. Cassava (manioc). 2020. URL: https://plantvillage.psu.edu/topics/cassava-manioc/infos.
5. Christopher A. K-Nearest Neighbor. 2021. URL: https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4.
6. The Math Behind KNN. 2021. URL: https://ai.plainenglish.io/the-math-behind-knn-7883aa8e314c
7. Воронцов К. В. Лекции по методу опорных векторов. 2017. URL: http://www.ccas.ru/voron/download/SVM.pdf
8. Classification Using Neural Networks. *Towards Data Science*. 2019. URL: https://towardsdatascience.com/classification-using-neural-networks-b8e98f3a904f.
9. Convolutional Neural Networks. *IBM Cloud Education*. 2020. URL: https://www.ibm.com/cloud/learn/convolutional-neural-networks.
10. Глубокое обучение (Deep Learning): краткий туториал. 2018. URL: https://neurohive.io/ru/osnovy-data-science/glubokoe-obuchenie-deep-learning-kratkij-tutorial/.
11. Convolutional Neural Networks. 2020. URL: https://www.ibm.com/cloud/learn/convolutional-neural-networks.

12. Mujtaba A. What is Rectified Linear Unit (ReLU). *Introduction to ReLU Activation Function*. 2020. URL: https://www.mygreatlearning.com/blog/relu-activation-function/#sh3.
13. Wood T. What is the Softmax Function. *Wood*. URL: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer.
14. Новые архитектуры нейросетей. 2020. URL: https://habr.com/ru/post/498168/.
15. VGG Very Deep Convolutional Networks (VGGNet) – What you need to know. 2021. URL: https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/.
16. Deep Residual Networks (ResNet, ResNet50) – Guide in 2021. 2021. URL: https://viso.ai/deep-learning/resnet-residual-neural-network/.
17. Рашид Т. Создаем нейронную сеть. Санкт-Петербург: Альфа-книга, 2017. 272 с.
18. Згуровський М. З. Основы вычислительного интеллекта. Київ: Наукова думка, 2013. 406 с
19. Martinez-Chico M. Cloud classification in a mediterranean location using radiation data and sky images. *Energy*. 2011. Vol.36. P. 4055–4062..

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
# %% [markdown]
# # About this notebook
# - PyTorch resnext50_32x4d starter code
# - StratifiedKFold 5 folds
#
# If this notebook is helpful, feel free to upvote :)

# %% [markdown]
# # Data Loading

# %% [code]
import os

import pandas as pd

from matplotlib import pyplot as plt
import seaborn as sns

# %% [code]
os.listdir('../input/cassava-leaf-disease-classification')

# %% [code]
train = pd.read_csv('../input/cassava-leaf-disease-classification/train.csv')
test = pd.read_csv('../input/cassava-leaf-disease-classification/sample_submission.csv')
label_map = pd.read_json('../input/cassava-leaf-disease-classification/label_num_to_disease_map.json',
```

```
                    orient='index')
display(train.head())
display(test.head())
display(label_map)

# %% [code]
sns.distplot(train['label'], kde=False)

# %% [markdown]
# # Directory settings

# %% [code]
# ====================================================
# Directory settings
# ====================================================
import os

OUTPUT_DIR = './'
if not os.path.exists(OUTPUT_DIR):
    os.makedirs(OUTPUT_DIR)

TRAIN_PATH = '../input/cassava-leaf-disease-classification/train_images'
TEST_PATH = '../input/cassava-leaf-disease-classification/test_images'

# %% [markdown]
# # CFG

# %% [code]
# ====================================================
# CFG
```

```
# ====================================================
class CFG:
    debug=False
    apex=False
    print_freq=100
    num_workers=4
    model_name='resnext50_32x4d'
    size=256
    scheduler='CosineAnnealingWarmRestarts'        # ['ReduceLROnPlateau',
'CosineAnnealingLR', 'CosineAnnealingWarmRestarts']
    epochs=10
    #factor=0.2 # ReduceLROnPlateau
    #patience=4 # ReduceLROnPlateau
    #eps=1e-6 # ReduceLROnPlateau
    #T_max=10 # CosineAnnealingLR
    T_0=10 # CosineAnnealingWarmRestarts
    lr=1e-4
    min_lr=1e-6
    batch_size=32
    weight_decay=1e-6
    gradient_accumulation_steps=1
    max_grad_norm=1000
    seed=42
    target_size=5
    target_col='label'
    n_fold=5
    trn_fold=[0, 1, 2, 3, 4]
    train=True
    inference=False
```

```
if CFG.debug:
    CFG.epochs = 1
    train = train.sample(n=1000, random_state=CFG.seed).reset_index(drop=True)

# %% [markdown]
# # Library

# %% [code]
# ====================================================
# Library
# ====================================================
import sys
sys.path.append('../input/pytorch-image-models/pytorch-image-models-master')

import os
import math
import time
import random
import shutil
from pathlib import Path
from contextlib import contextmanager
from collections import defaultdict, Counter

import scipy as sp
import numpy as np
import pandas as pd

from sklearn import preprocessing
from sklearn.metrics import accuracy_score
from sklearn.model_selection import StratifiedKFold
```

```
from tqdm.auto import tqdm
from functools import partial

import cv2
from PIL import Image

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.optim import Adam, SGD
import torchvision.models as models
from torch.nn.parameter import Parameter
from torch.utils.data import DataLoader, Dataset
from torch.optim.lr_scheduler import CosineAnnealingWarmRestarts, CosineAnnealingLR, ReduceLROnPlateau

from albumentations import (
    Compose, OneOf, Normalize, Resize, RandomResizedCrop, RandomCrop, HorizontalFlip, VerticalFlip,
    RandomBrightness, RandomContrast, RandomBrightnessContrast, Rotate, ShiftScaleRotate, Cutout,
    IAAAdditiveGaussianNoise, Transpose
    )
from albumentations.pytorch import ToTensorV2
from albumentations import ImageOnlyTransform

import timm

import warnings
```

```
warnings.filterwarnings('ignore')

if CFG.apex:
    from apex import amp

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# %% [markdown]
# # Utils

# %% [code]
# ====================================================
# Utils
# ====================================================
def get_score(y_true, y_pred):
    return accuracy_score(y_true, y_pred)


@contextmanager
def timer(name):
    t0 = time.time()
    LOGGER.info(f'[{name}] start')
    yield
    LOGGER.info(f'[{name}] done in {time.time() - t0:.0f} s.')


def init_logger(log_file=OUTPUT_DIR+'train.log'):
    from logging import getLogger, INFO, FileHandler,  Formatter,  StreamHandler
    logger = getLogger(__name__)
    logger.setLevel(INFO)
```

```
    handler1 = StreamHandler()
    handler1.setFormatter(Formatter("%(message)s"))
    handler2 = FileHandler(filename=log_file)
    handler2.setFormatter(Formatter("%(message)s"))
    logger.addHandler(handler1)
    logger.addHandler(handler2)
    return logger

LOGGER = init_logger()



def seed_torch(seed=42):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True

seed_torch(seed=CFG.seed)

# %% [markdown]
# # CV split

# %% [code]
folds = train.copy()
Fold = StratifiedKFold(n_splits=CFG.n_fold, shuffle=True,
random_state=CFG.seed)
for n, (train_index, val_index) in enumerate(Fold.split(folds,
folds[CFG.target_col])):
```

```
    folds.loc[val_index, 'fold'] = int(n)
folds['fold'] = folds['fold'].astype(int)
print(folds.groupby(['fold', CFG.target_col]).size())

# %% [markdown]
# # Dataset

# %% [code]
# ====================================================
# Dataset
# ====================================================
class TrainDataset(Dataset):
    def __init__(self, df, transform=None):
        self.df = df
        self.file_names = df['image_id'].values
        self.labels = df['label'].values
        self.transform = transform

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        file_name = self.file_names[idx]
        file_path = f'{TRAIN_PATH}/{file_name}'
        image = cv2.imread(file_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        if self.transform:
            augmented = self.transform(image=image)
            image = augmented['image']
        label = torch.tensor(self.labels[idx]).long()
```

```
        return image, label


class TestDataset(Dataset):
    def __init__(self, df, transform=None):
        self.df = df
        self.file_names = df['image_id'].values
        self.transform = transform

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        file_name = self.file_names[idx]
        file_path = f'{TEST_PATH}/{file_name}'
        image = cv2.imread(file_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        if self.transform:
            augmented = self.transform(image=image)
            image = augmented['image']
        return image

# %% [code]
train_dataset = TrainDataset(train, transform=None)

for i in range(1):
    image, label = train_dataset[i]
    plt.imshow(image)
    plt.title(f'label: {label}')
    plt.show()
```

```
# %% [markdown]
# # Transforms

# %% [code]
# ====================================================
# Transforms
# ====================================================
def get_transforms(*, data):

    if data == 'train':
        return Compose([
            #Resize(CFG.size, CFG.size),
            RandomResizedCrop(CFG.size, CFG.size),
            Transpose(p=0.5),
            HorizontalFlip(p=0.5),
            VerticalFlip(p=0.5),
            ShiftScaleRotate(p=0.5),
            Normalize(
                mean=[0.485, 0.456, 0.406],
                std=[0.229, 0.224, 0.225],
            ),
            ToTensorV2(),
        ])

    elif data == 'valid':
        return Compose([
            Resize(CFG.size, CFG.size),
            Normalize(
                mean=[0.485, 0.456, 0.406],
```

```
            std=[0.229, 0.224, 0.225],
        ),
        ToTensorV2(),
    ])

# %% [code]
train_dataset = TrainDataset(train, transform=get_transforms(data='train'))

for i in range(1):
    image, label = train_dataset[i]
    plt.imshow(image[0])
    plt.title(f'label: {label}')
    plt.show()

# %% [markdown]
# # MODEL

# %% [code]
# ====================================================
# MODEL
# ====================================================
class CustomResNext(nn.Module):
    def __init__(self, model_name='resnext50_32x4d', pretrained=False):
        super().__init__()
        self.model = timm.create_model(model_name, pretrained=pretrained)
        n_features = self.model.fc.in_features
        self.model.fc = nn.Linear(n_features, CFG.target_size)

    def forward(self, x):
        x = self.model(x)
```

```
        return x

# %% [code]
model = CustomResNext(model_name=CFG.model_name, pretrained=False)
train_dataset = TrainDataset(train, transform=get_transforms(data='train'))
train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True,
                          num_workers=4, pin_memory=True, drop_last=True)

for image, label in train_loader:
    output = model(image)
    print(output)
    break

# %% [markdown]
# # Helper functions

# %% [code]
# ====================================================
# Helper functions
# ====================================================
class AverageMeter(object):
    """Computes and stores the average and current value"""
    def __init__(self):
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0
```

```
    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count


def asMinutes(s):
    m = math.floor(s / 60)
    s -= m * 60
    return '%dm %ds' % (m, s)


def timeSince(since, percent):
    now = time.time()
    s = now - since
    es = s / (percent)
    rs = es - s
    return '%s (remain %s)' % (asMinutes(s), asMinutes(rs))


def train_fn(train_loader, model, criterion, optimizer, epoch, scheduler, device):
    batch_time = AverageMeter()
    data_time = AverageMeter()
    losses = AverageMeter()
    scores = AverageMeter()
    # switch to train mode
    model.train()
    start = end = time.time()
```

```
    global_step = 0
    for step, (images, labels) in enumerate(train_loader):
        # measure data loading time
        data_time.update(time.time() - end)
        images = images.to(device)
        labels = labels.to(device)
        batch_size = labels.size(0)
        y_preds = model(images)
        loss = criterion(y_preds, labels)
        # record loss
        losses.update(loss.item(), batch_size)
        if CFG.gradient_accumulation_steps > 1:
            loss = loss / CFG.gradient_accumulation_steps
        if CFG.apex:
            with amp.scale_loss(loss, optimizer) as scaled_loss:
                scaled_loss.backward()
        else:
            loss.backward()
        grad_norm = torch.nn.utils.clip_grad_norm_(model.parameters(),
CFG.max_grad_norm)
        if (step + 1) % CFG.gradient_accumulation_steps == 0:
            optimizer.step()
            optimizer.zero_grad()
            global_step += 1
        # measure elapsed time
        batch_time.update(time.time() - end)
        end = time.time()
        if step % CFG.print_freq == 0 or step == (len(train_loader)-1):
            print('Epoch: [{0}][{1}/{2}] '
                  'Data {data_time.val:.3f} ({data_time.avg:.3f}) '
```

```
                'Elapsed {remain:s} '
                'Loss: {loss.val:.4f}({loss.avg:.4f}) '
                'Grad: {grad_norm:.4f}  '
                #'LR: {lr:.6f}  '
                .format(
                 epoch+1, step, len(train_loader), batch_time=batch_time,
                 data_time=data_time, loss=losses,
                 remain=timeSince(start, float(step+1)/len(train_loader)),
                 grad_norm=grad_norm,
                 #lr=scheduler.get_lr()[0],
                 ))
    return losses.avg


def valid_fn(valid_loader, model, criterion, device):
    batch_time = AverageMeter()
    data_time = AverageMeter()
    losses = AverageMeter()
    scores = AverageMeter()
    # switch to evaluation mode
    model.eval()
    preds = []
    start = end = time.time()
    for step, (images, labels) in enumerate(valid_loader):
        # measure data loading time
        data_time.update(time.time() - end)
        images = images.to(device)
        labels = labels.to(device)
        batch_size = labels.size(0)
        # compute loss
```

```
        with torch.no_grad():
            y_preds = model(images)
        loss = criterion(y_preds, labels)
        losses.update(loss.item(), batch_size)
        # record accuracy
        preds.append(y_preds.softmax(1).to('cpu').numpy())
        if CFG.gradient_accumulation_steps > 1:
            loss = loss / CFG.gradient_accumulation_steps
        # measure elapsed time
        batch_time.update(time.time() - end)
        end = time.time()
        if step % CFG.print_freq == 0 or step == (len(valid_loader)-1):
            print('EVAL: [{0}/{1}] '
                  'Data {data_time.val:.3f} ({data_time.avg:.3f}) '
                  'Elapsed {remain:s} '
                  'Loss: {loss.val:.4f}({loss.avg:.4f}) '
                  .format(
                   step, len(valid_loader), batch_time=batch_time,
                   data_time=data_time, loss=losses,
                   remain=timeSince(start, float(step+1)/len(valid_loader)),
                   ))
    predictions = np.concatenate(preds)
    return losses.avg, predictions


def inference(model, states, test_loader, device):
    model.to(device)
    tk0 = tqdm(enumerate(test_loader), total=len(test_loader))
    probs = []
    for i, (images) in tk0:
```

```
        images = images.to(device)
        avg_preds = []
        for state in states:
            model.load_state_dict(state['model'])
            model.eval()
            with torch.no_grad():
                y_preds = model(images)
            avg_preds.append(y_preds.softmax(1).to('cpu').numpy())
        avg_preds = np.mean(avg_preds, axis=0)
        probs.append(avg_preds)
    probs = np.concatenate(probs)
    return probs

# %% [markdown]
# # Train loop

# %% [code]
# ====================================================
# Train loop
# ====================================================
def train_loop(folds, fold):

    LOGGER.info(f"========== fold: {fold} training ==========")

    # ====================================================
    # loader
    # ====================================================
    trn_idx = folds[folds['fold'] != fold].index
    val_idx = folds[folds['fold'] == fold].index
```

```
    train_folds = folds.loc[trn_idx].reset_index(drop=True)
    valid_folds = folds.loc[val_idx].reset_index(drop=True)

    train_dataset = TrainDataset(train_folds,
                                 transform=get_transforms(data='train'))
    valid_dataset = TrainDataset(valid_folds,
                                 transform=get_transforms(data='valid'))

    train_loader = DataLoader(train_dataset,
                              batch_size=CFG.batch_size,
                              shuffle=True,
                              num_workers=CFG.num_workers,          pin_memory=True,
drop_last=True)
    valid_loader = DataLoader(valid_dataset,
                              batch_size=CFG.batch_size,
                              shuffle=False,
                              num_workers=CFG.num_workers,          pin_memory=True,
drop_last=False)

    # ====================================================
    # scheduler
    # ====================================================
    def get_scheduler(optimizer):
        if CFG.scheduler=='ReduceLROnPlateau':
            scheduler = ReduceLROnPlateau(optimizer, mode='min',
factor=CFG.factor, patience=CFG.patience, verbose=True, eps=CFG.eps)
        elif CFG.scheduler=='CosineAnnealingLR':
            scheduler = CosineAnnealingLR(optimizer, T_max=CFG.T_max,
eta_min=CFG.min_lr, last_epoch=-1)
        elif CFG.scheduler=='CosineAnnealingWarmRestarts':
```

```
        scheduler = CosineAnnealingWarmRestarts(optimizer, T_0=CFG.T_0, T_mult=1, eta_min=CFG.min_lr, last_epoch=-1)
        return scheduler

    # ====================================================
    # model & optimizer
    # ====================================================
    model = CustomResNext(CFG.model_name, pretrained=True)
    model.to(device)

    optimizer = Adam(model.parameters(), lr=CFG.lr, weight_decay=CFG.weight_decay, amsgrad=False)
    scheduler = get_scheduler(optimizer)

    # ====================================================
    # apex
    # ====================================================
    if CFG.apex:
        model, optimizer = amp.initialize(model, optimizer, opt_level='O1', verbosity=0)

    # ====================================================
    # loop
    # ====================================================
    criterion = nn.CrossEntropyLoss()

    best_score = 0.
    best_loss = np.inf

    for epoch in range(CFG.epochs):
```

```
    start_time = time.time()

    # train
    avg_loss = train_fn(train_loader, model, criterion, optimizer, epoch, scheduler,
device)

    # eval
    avg_val_loss, preds = valid_fn(valid_loader, model, criterion, device)
    valid_labels = valid_folds[CFG.target_col].values

    if isinstance(scheduler, ReduceLROnPlateau):
        scheduler.step(avg_val_loss)
    elif isinstance(scheduler, CosineAnnealingLR):
        scheduler.step()
    elif isinstance(scheduler, CosineAnnealingWarmRestarts):
        scheduler.step()

    # scoring
    score = get_score(valid_labels, preds.argmax(1))

    elapsed = time.time() - start_time

    LOGGER.info(f'Epoch {epoch+1} - avg_train_loss: {avg_loss:.4f}
avg_val_loss: {avg_val_loss:.4f}  time: {elapsed:.0f}s')
    LOGGER.info(f'Epoch {epoch+1} - Accuracy: {score}')

    if score > best_score:
        best_score = score
```

```
            LOGGER.info(f'Epoch {epoch+1} - Save Best Score: {best_score:.4f} Model')
            torch.save({'model': model.state_dict(),
                        'preds': preds},
                        OUTPUT_DIR+f'{CFG.model_name}_fold{fold}_best.pth')

    check_point = torch.load(OUTPUT_DIR+f'{CFG.model_name}_fold{fold}_best.pth')
    valid_folds[[str(c) for c in range(5)]] = check_point['preds']
    valid_folds['preds'] = check_point['preds'].argmax(1)

    return valid_folds
# %% [code]
# ====================================================
# main
# ====================================================
def main():
    """
    Prepare: 1.train  2.test  3.submission  4.folds
    """
    def get_result(result_df):
        preds = result_df['preds'].values
        labels = result_df[CFG.target_col].values
        score = get_score(labels, preds)
        LOGGER.info(f'Score: {score:<.5f}')

    if CFG.train:
        # train
        oof_df = pd.DataFrame()
        for fold in range(CFG.n_fold):
```

```
        if fold in CFG.trn_fold:
            _oof_df = train_loop(folds, fold)
            oof_df = pd.concat([oof_df, _oof_df])
            LOGGER.info(f"========== fold: {fold} result ==========")
            get_result(_oof_df)
    # CV result
    LOGGER.info(f"========== CV ==========")
    get_result(oof_df)
    # save result
    oof_df.to_csv(OUTPUT_DIR+'oof_df.csv', index=False)
    if CFG.inference:
    # inference
    model = CustomResNext(CFG.model_name, pretrained=False)
    states = [torch.load(OUTPUT_DIR+f'{CFG.model_name}_fold{fold}_best.pth') for fold in CFG.trn_fold]
    test_dataset = TestDataset(test, transform=get_transforms(data='valid'))
    test_loader = DataLoader(test_dataset, batch_size=CFG.batch_size, shuffle=False,
                             num_workers=CFG.num_workers, pin_memory=True)
    predictions = inference(model, states, test_loader, device)
    # submission
    test['label'] = predictions.argmax(1)
    test[['image_id', 'label']].to_csv(OUTPUT_DIR+'submission.csv', index=False)

# %% [code]
if __name__ == '__main__':
    main()
```