НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

Завідувач кафедри

_____ О.Л. Тимощук

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

на тему: «Система розпізнавання патологій рослин»

Виконав:
студент IV курсу, групи КА-71
Оксюта Ірина Миколаївна _____

Керівник:
доцент, к.т.н. Дідковська М.В. _____

Консультант з економічного розділу:
доцент, к.е.н. Рощина Н.В. _____

Консультант з нормоконтролю:
доцент, к.т.н. Коваленко А.Є. _____

Рецензент:
доцент, к.т.н. Заболотня Т.М. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2021 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ О.Л. Тимощук

«___»_____20__ р.

ЗАВДАННЯ

на дипломну роботу студенту

Оксюти Ірини Миколаївни

1. Тема роботи «Система розпізнавання патологій рослин», керівник роботи доцент, канд. тех. наук Дідковська М.В., затверджені наказом по університету від «26» травня 2021 р. № 1344-с

2. Термін подання студентом роботи 7 червня 2021р.

3. Вихідні дані до роботи: модель розпізнавання, система розпізнавання листяних патологій рослин

4. Зміст роботи: дослідження предметної області, огляд математичних підходів до задачі розпізнавання, застосування згорткових нейронних мереж для поставленої задачі, практичні результати та їх аналіз.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В., доцент		

7. Дата видачі завдання: 12.04.2021 _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Затвердження теми ДР	12.04.2021-20.04.2021	виконано
2	Ознайомлення зі структурою БДР згідно з Положенням про державну атестацію студентів НТУУ «КПІ ім. І. Сікорського»	12.04.2020-27.04.2021	виконано
3	Ознайомлення з ДСТУ 3008-95 та стандарти ЄСПД	20.04.2021-27.04.2021	виконано
4	Проведення дослідження за темою БДР під керівництвом керівника	20.04.2021-04.05.2021	виконано
5	Завершення роботи над першим варіантом частини БДР	05.05.2021-12.05.2021	виконано
6	Проведення роботи над експериментальною частиною БДР	05.05.2021-16.05.2021	виконано
7	Проведення роботи над програмним продуктом	16.05.2021-25.05.2021	виконано
8	Оформлення БДР та аналіз отриманих результатів	16.05.2021-25.05.2021	виконано

Студент І.М. Оксюта

Керівник М.В. Дідковська

РЕФЕРАТ

Дипломна робота: 136 с., 48 рис., 7 табл., 2 додатків, 17 джерел.

РОЗПІЗНАВАННЯ ОБРАЗІВ, КОМП'ЮТЕРНИЙ ЗІР, БАГАТОМІТКОВА КЛАСИФІКАЦІЯ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ.

Предмет дослідження: алгоритми розпізнавання зображень.

Об'єкт дослідження: відкритий набір даних, який включає зображення різних листяних патологій яблунь.

Мета дослідження: проаналізувати існуючі моделі розпізнавання, розробити власну систему розпізнавання патологій рослин на прикладі позакореневих захворювань яблучних дерев.

Використані моделі: у програмній реалізації було використано згорткові нейронні мережі.

Актуальність роботи зумовлена автоматизацією виробничих процесів на всіх ланках. Система розпізнавання листяних патологій дозволяє зменшити витрати ресурсів на ручну діагностику насаджень, проводити своєчасні обробки, запобігати поширенню хвороб.

Отриманні результати: побудована система розпізнавання патологій рослин, що може надавати прогноз відносно окремих категорій листяних захворювань з прийнятною точністю.

В рамках подальшого дослідження пропонується підвищувати точність отриманої моделей, розширювати вхідний набір даних, інтегрувати систему в готові продукти для роботи в реальному часі. Також можливе розширення сукупності міток захворювань, або побудова відповідних моделей для інших рослин.

ABSTRACT

Thesis: 136 p., 48 fig., 7 tabl., 2 appendices, 17 sources.

PATTERN RECOGNITION, COMPUTER VISION, MULTI-LABEL CLASSIFICATION, CONVOLUTIONAL NEURAL NETWORKS.

Subject of research: image recognition algorithms.

Object of research: an open dataset that includes images of apple`s leaf diseases.

The purpose of the work: to analyze existing recognition models, to develop own plant pathology recognition system on the example of the apple`s foliar diseases.

Used models: convolutional neural networks were used in the software implementation.

The relevance of the work is due to the automation of technology processes at all stages. Plant pathology recognition system allows to reduce the loss of resources for manual diagnostics of plantations, to carry out timely treatments, to prevent the spread of diseases.

The results obtained: a plant pathology recognition system was built, which can provide a forecast for certain categories of foliar diseases with acceptable accuracy.

As part of further research, it is proposed to increase the accuracy of the obtained models, expand the input dataset, integrate the system into ready-made products for real-time operation. It is also possible to expand the set of diseases labels, or build appropriate models for other plants.

ЗМІСТ

ВСТУП

Яблука - одна з найважливіших плодових культур помірного клімату. Позакореневі (листяні) патології становлять основну загрозу загальній продуктивності садових господарств. На даний час вже з'явились ефективні препарати для боротьби з різного роду ураженнями. Використовуючи їх можна суттєво вплинути на якість фінального продукту та запобігти поширенню хвороб. Але в даній справі дуже важливу роль грає своєчасність обробки. А для цього необхідно точно виявити захворювання, особливо на початкових етапах.

Сучасний процес діагностики захворювань у яблуневих садах базується на ручній розвідці, яка виконується людьми та займає багато часу і коштів. У зв'язку з цим стоїть задача розробки більш ефективних методів дослідження, насамперед систем з використанням моделей комп'ютерного зору. Це дозволить масштабувати діагностику на великі господарства, в розпорядженні яких не один гектар саду. Також, результати роботи можна застосувати до інших плодових культур, при цьому враховуючи специфіку їхніх патологій.

Основною метою цієї роботи є розробка моделей, заснованих на машинному навчанні, для точної класифікації окремих зображень листя до певної категорії хвороби, а також для виявлення окремого захворювання із множинних симптомів на одному зображенні. Вихідні дані для роботи являють собою сукупність з майже 19000 високоякісних RGB зображень позакореневих захворювань яблунь, розмічених експертами на окремі категорії хвороб.

Перший розділ дипломної роботи присвячений аналізу задачі розпізнавання. Детально розглянуто актуальність побудови конкретної системи, існуючі підходи до вирішення подібної проблеми. Виконано початковий аналіз вихідних даних та проведена формалізація постановки цієї задачі. У другому розділі описано математичні основи для задач комп'ютерного зору, визначено засади проектування та побудови відповідних моделей. Виконано огляд існуючих рішень та порівняння результатів їхньої роботи. У третьому розділі

проведено аналіз та підготовку вхідних даних до процесу навчання, описано архітектуру побудованої системи, виконано аналіз отриманих результатів та налаштування параметрів моделі. Також проведено експеримент з використанням нових тестових даних для оцінки якості створеного програмного продукту. У четвертому розділі приведений функціонально-вартісний аналіз роботи.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

У сучасному світі досить гостро стоїть питання продовольства. В даній роботі розглянуто одну з основних плодових культур – яблука. Вони займають третє місце по обсягами споживання фруктів у світі, після бананів та кавунів. Лідируючими країнами-виробниками є Китай, Сполучені Штати Америки, Туреччина. Щороку фермери намагаються збільшити обсяги виробництва та загалом свій прибуток. Проте існує низка факторів які можуть суттєво вплинути на виробничі плани господарств. Несприятливі погодні умови, техногенні та нетехногенні катаклізми, різноманітні патології самих рослин. Але якщо на клімат досить майже неможливо вплинути, то уберегти насадження від хвороб - справа цілком реальна в наш час. Для всіх господарств, особливо масштабних, важливим є своєчасна діагностика захворювань та ефективне втручання в життєвий цикл саду.

Наразі використання людського ресурсу на всіх етапах виробництва є досить коштовним та займає багато часу. Все більше компаній впроваджують нові системи автоматизації для збору, обробки та упаковки продукції. Активно застосовуються системи розпізнавання які використовують технології комп’ютерного зору, наприклад для відбраковування пошкоджених плодів що рухаються по конвеєру. Але догляд та спостереження за станом саду в теперішній час також потребує нових підходів та автоматизації.

Незважаючи на те, що моделі, що базуються на комп’ютерному зорі, обіцяють ідентифікувати хвороби рослин, є деякі обмеження, які потрібно вирішити. Великі варіації візуальних симптомів однієї хвороби у різних сортів яблуні або нових сортів, що виникли під час вирощування, є основними проблемами для ідентифікації хвороб на основі комп'ютерного зору. Ці варіації

виникають внаслідок відмінностей у природних середовищах та середовищах, що фіксують зображення, наприклад, колір листя та морфологія листя, вік інфікованих тканин, нерівномірний фон зображення та різне ступінь освітленості тощо.

Актуальність даного дослідження полягає в запиті фермерів до розробки нових засобів розпізнавання захворювань рослин у зв'язку з автоматизацією виробничих процесів. Також отриманий підхід до вирішення проблеми можна застосувати і до інших споживчих культур, враховуючи специфіку їхніх патологій. Або можна покращити дану систему для конкретної рослини(яблуні), шляхом розширення навчального набору даних та збільшенням кількості класів хвороб.

1.2 Аналіз існуючих підходів

Оскільки окрім набору зображень, ми також маємо мітки хвороб до кожного з них, то перед нами класична задача supervised learning, Необхідно натренувати якісну модель для прогнозування міток на нових даних. Складність роботи полягає утому що окремому зображенню може відповідати не одна патологія, а деяка їх сукупність. Для подібних задач комп'ютерного зору (multi-label classification) використовують описані далі підходи.

1.2.1 Класичні моделі класифікації

Перед початком роботи з моделлю необхідно для кожного зображення із набору сформувати свій набір характерних ознак, опираючись на значення яких, в подальшому можна тренувати класифікатор(рис. 1.1). Можна виділити колір, фактуру та форму як про основні ознаки, проте вони не в достатній мірі унікально описують кожен об'єкт. Тому використовують так звані локальні ознаки, які отримують за допомогою функцій-дескрипторів. Вони кількісно

визначають локальні області зображення та особливі точки на всьому зображенні. Потім дескриптор перекодовує цю інформацію в ряд цифр, які виступають як свого роду «відбиток пальця», який можна використовувати для диференціації однієї ознаки від іншої. В ідеалі ця інформація буде незмінною при перетворенні зображення, тому ми можемо знайти ознаки, навіть якщо зображення певним чином змінено. Деякі з найбільш часто використовуваних локальних дескрипторів:

- SIFT (Scale Invariant Feature Transform)
- SURF (Speeded Up Robust Features)
- ORB (Oriented Fast and Rotated BRIEF)
- BRIEF (Binary Robust Independed Elementary Features)

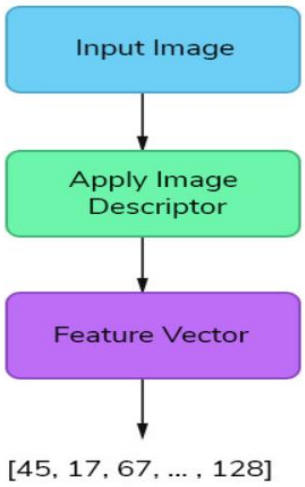


Рисунок 1.1 – Схема вилучення характерних ознак зображення

Також в подібних задачах досить часто використовують загальний дескриптор Histogram of Oriented Gradients(HOG). HOG використовують для зменшення кількості ознак, іншими словами: для зниження складності проблеми, зберігаючи якомога більше варіацій вихідного зображення. На кожному пікселі градієнт має величину та напрямок. Для кольорових зображень оцінюються абсолютні градієнти горизонтальних та вертикальних лінії, а також значення градієнту в кожній точці. Величина градієнта в пікселі - це максимум величини

градієнтів з трьох каналів, а кут - це кут, що відповідає максимальному градієнту. Так, для обчислення HOG, зображення ділиться на блоки, наприклад 8 на 8 обчислюється величина градієнта в заданій кількості напрямків. Наступним кроком є створення гістограми градієнтів для цих блоків 8 × 8. Гістограма містить 9 bin, що відповідають кутам 0, 20, 40 ... 160(рис. 1.2).

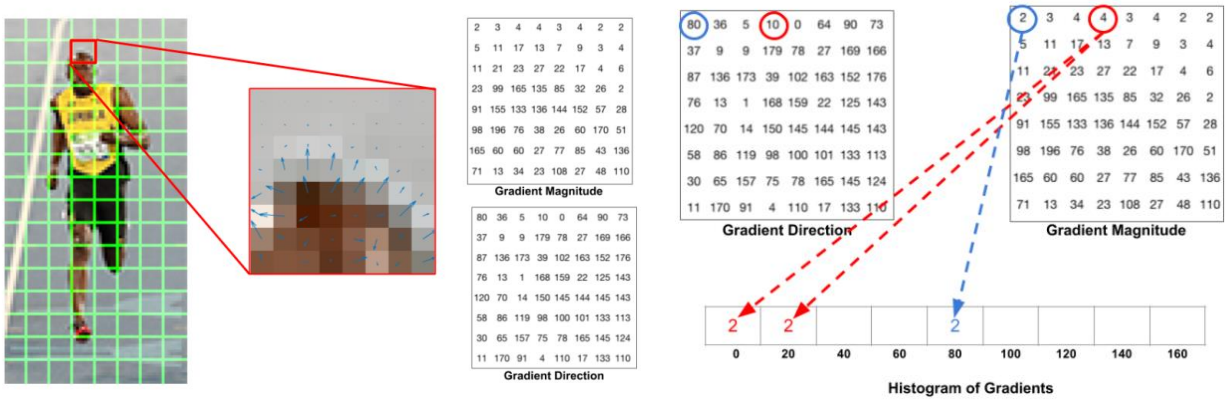


Рисунок 1.2 – Принцип роботи дескриптора HOG

Таким чином для кожного з блоків ми отримуємо унікальну гістограму, що містить найважливіші дані про зображення, і водночас зменшуємо кількість факторів для подальшого використання з моделлю навчання. Наприклад, як показано на рисунку 1.3.

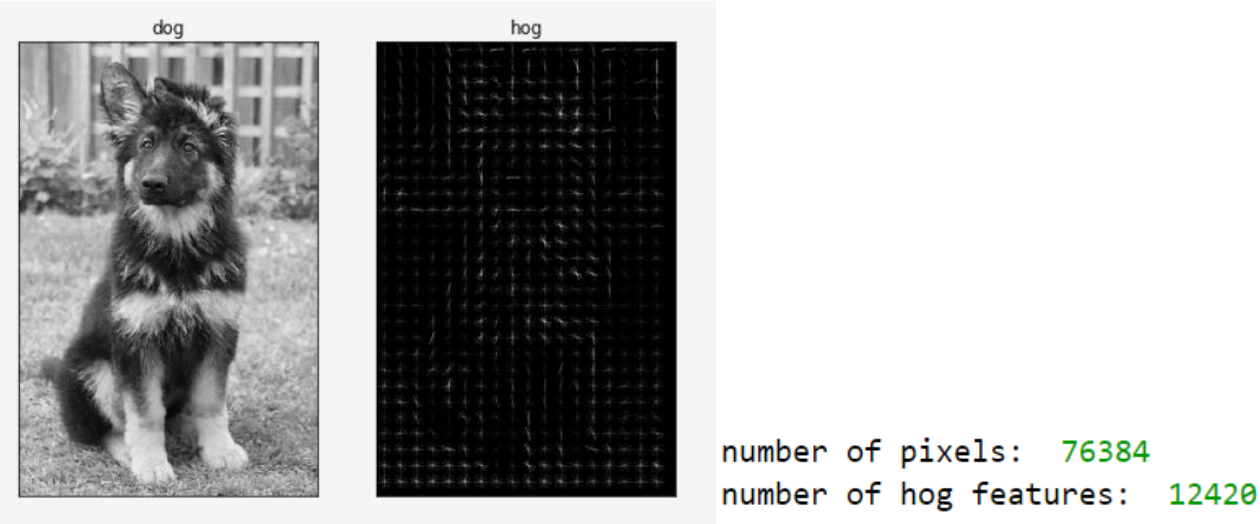


Рисунок 1.3 – Приклад роботи HOG для зменшення кількості ознак

Отже загальний підхід до вирішення поставленої задачі класичними методами класифікації можна описати наступним чином (рис.1.4): вхідне зображення проходить етап початкової обробки(зміна контрастності, кольору, розміру), потім застосовують дескриптори для вилучення ознак і вже на основі отриманих факторів проводиться навчання класифікатора.

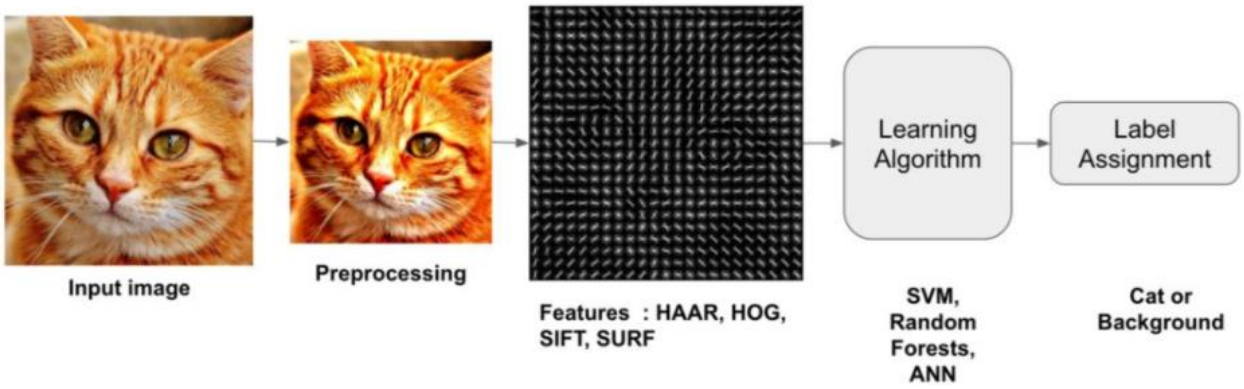


Рисунок 1.4 – Класичний алгоритм роботи для задачі класифікації зображення

В якості навчального алгоритму можуть виступати наступні моделі класифікації: Logistic Regression, Linear Discriminant Analysis, K-Nearest Neighbors, Decision Trees, Random Forests, Gaussian Naive Bayes, Support Vector Machine, Artificial Neural Network. Для попередньої обробки зображення та застосування дескрипторів найчастіше використовують open-source бібліотеку OpenCV.

1.2.2 Згорткові нейронні мережі

Convolutional Neural Networks (CNNs) – найпопулярніша модель нейронної мережі, яка використовується для вирішення проблеми класифікації зображень. Головна ідея CNN полягає в тому, що вона досить добре розуміє окремі області образу. Практична перевага даного рішення в тому, що менша кількість необхідних параметрів значно покращує час навчання, а також зменшує обсяг даних, потрібних для тренування моделі. Замість повністю підключеної мережі

ваг з кожного пікселя, моделі достатньо вагових коефіцієнтів з невеликого фрагменту зображення. Схему роботи показано на рисунку 1.5.

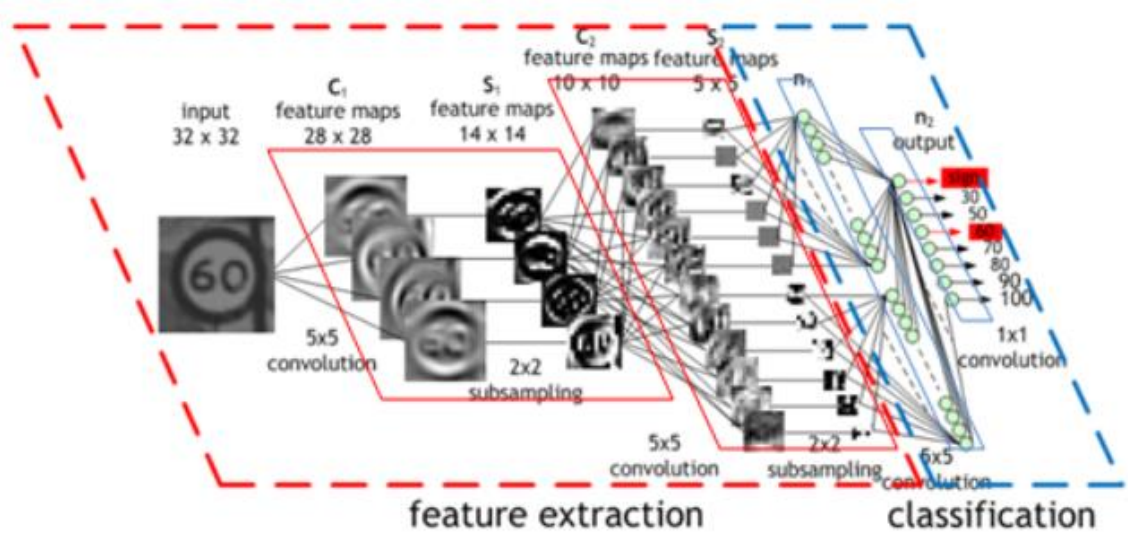


Рисунок 1.5– Загальний принцип роботи CNN

CNN складається з convolutional layers, pooling layers, activation function і fully connected layer. Все починається з шару згортки. У нього вводиться зображення (матриця зі значеннями пікселів). Далі обирається менша матриця, яка називається фільтром (або нейроном, або ядром). Потім фільтр виконує згортку, тобто рухається вздовж вхідного зображення. Завдання фільтра – помножити його значення на початкові значення пікселів. Всі ці множення підсумовуються. У підсумку виходить одне число. Оскільки фільтр зчитував зображення лише у верхньому лівому куті, він рухається все далі вправо на 1 одиницю, виконуючи подібну операцію. Після передачі фільтра по всіх положеннях виходить матриця, але менша за вхідну.

Мережа складається з декількох згорткових шарів, змішаних з функціями активації та пулінг-шарами. Коли зображення проходить через один шар згортки, вихід першого шару стає вхідним для другого шару. І це трапляється з кожним подальшим згортковим шаром. Передавальний шар додається після кожної операції згортки. Він містить функцію активації, яка вносить нелінійні властивості. Без цієї властивості мережа не буде достатньо інтенсивною і не зможе змоделювати змінну відповіді (як мітку класу).

Далі слідує пулінг-шар. Він працює з шириною та висотою зображення та виконує для них операцію зменшення дискретизації. В результаті зменшується обсяг зображення. Це означає, що якщо деякі ознаки (наприклад, межі) вже були ідентифіковані в попередній операції згортки, то детальне зображення більше не потрібно для подальшої обробки, і воно стискається.

Після завершення ряду згорткових та пулінг шарів необхідно прикріпити fully connected шар. Цей рівень приймає вихідну інформацію із згорткових мереж. Результатом роботи даного шару є N-мірний вектор, де N – кількість можливих класів. Повна схема роботи представлена на рисунку 1.6.

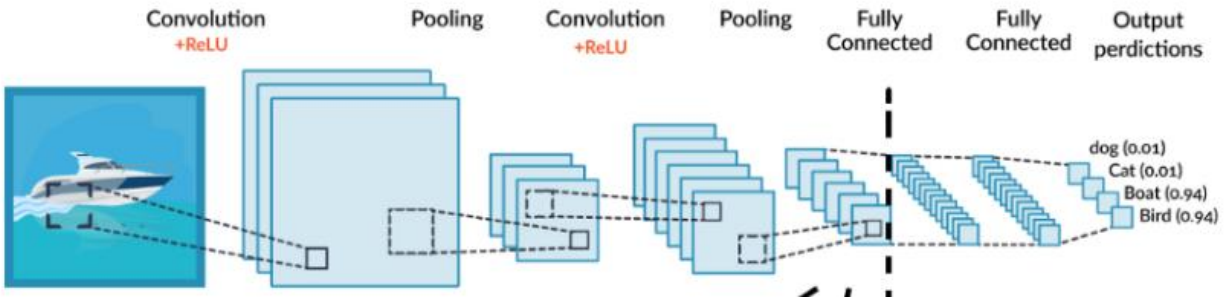


Рисунок 1.6 – Розгорнута схема роботи CNN класифікатора, тут ReLu – функція активації

Серед найвідоміших CNN можна виділити наступні: LeNet, AlexNet, VGG, GoogLeNet, ResNet, Unet. Активно використовуються такі бібліотеки як Tensorflow та PyTorch, які містять вже готові до використання мережі.

1.3 Опис даних для аналізу

Набір даних сформований для змагання «Plant Pathology 2021 FGVC8» на ресурсі http://kaggle.com. Навчальна вибірка містить більш ніж 18 тисяч високоякісних зображень, що анотовані на окремі категорії хвороб відповідними експертами. Цей датасет відображає реальні польові сценарії, представляючи неоднорідні фони зображень листків, зроблених на різних стадіях зрілості та в різний час доби за різних налаштувань фокусної камери. Файли в форматі .jpg

мають велику роздільну здатність (2672×4000), що може суттєво збільшити час обробки кожного окремого зображення. Набір складається з train-файлів, які також містять train.csv з описом міток для кожного зображення. В test-файлах міститься нерозмічений набір фотографій, який не бере участь в навчанні, за допомогою нього здійснюється перевірка якості прогнозу.

В train-наборі представлено 12 класів, проте можна помітити що різних категорій хвороб всього 6 (healthy, scab, rust, frog_eye_leaf_spot, complex, powdery_mildew), інші мітки є композицією патологій. Таким чином розподіл є доволі нерівномірним(рис.1.7). Приклад екземплярів з набору для різних категорій показано на рисунку 1.8.

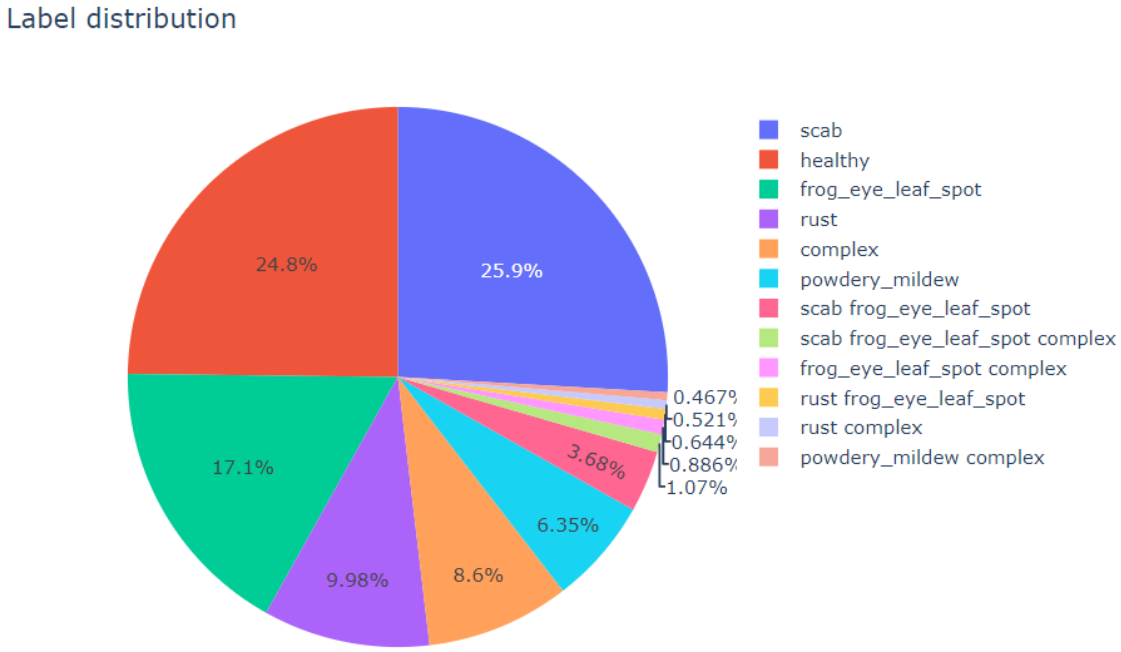


Рисунок 1.7 – Розподіл міток класів у тренувальному наборі

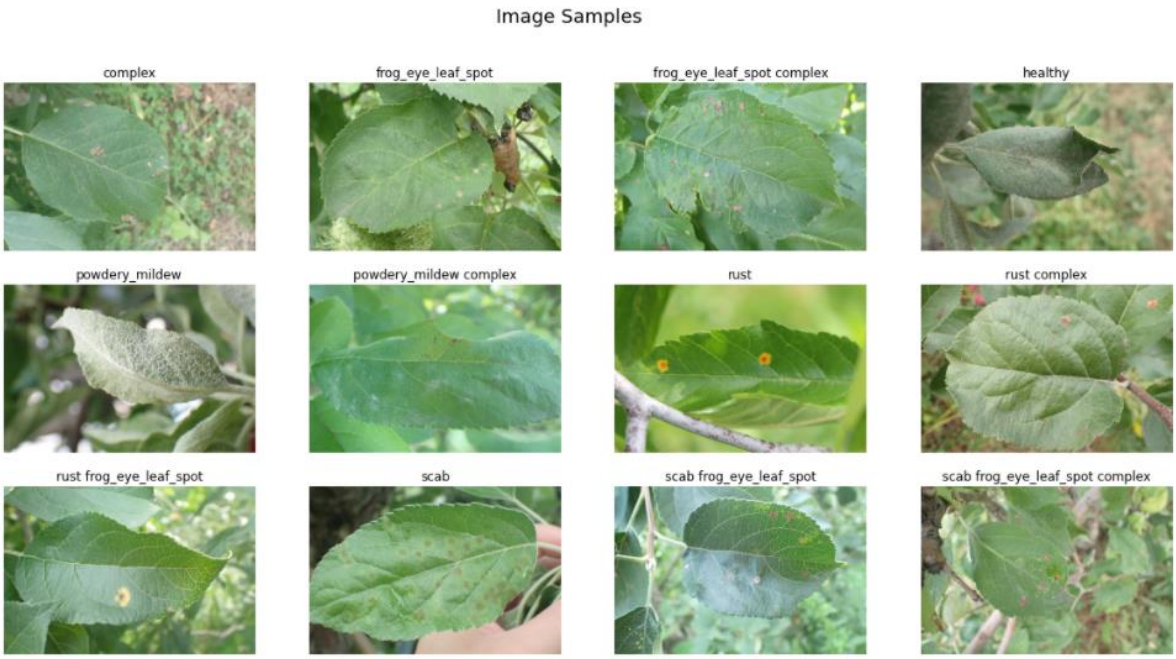


Рисунок 1.8 – Зображення окремих категорій хвороб

1.4 Формалізація задачі розпізнавання патологій

Постановка задачі класифікації зображень виглядає так: враховуючи набір зображень, які всі позначені однією чи декількома категоріями, нам пропонується передбачити ці категорії для нового набору тестових зображень та виміряти точність передбачень. З цим завданням пов’язано цілий ряд проблем, включаючи зміну точки зору, зміну масштабу, варіації в межах класу, деформацію зображення, оклюзію зображення, умови освітлення, безлад у фоновому режимі тощо. Незважаючи на те що вхідний набір даних досить різноманітний, подібні деталі можуть суттєво вплинути на якість прогнозу. Тому в першу чергу стоїть завдання добре підготовити дані до подальшого навчання.

Попередня підготовка навчальної вибірки насамперед повинна вирішити питання нерівномірності розподілу міток класів. Для покращення прогнозування можливо треба штучно збільшити кількість зображень, змінюючи нахил, контрастність, масштаб, тощо. Також важливо підготовити файли такого великого розміру до подальшої обробки, намагаючись оптимізувати час роботи графічного процесора.

На етапі навчання необхідно обрати модель яка найкращим чином описує навчальні дані та дає досить високу точність прогнозу для нових зображень. Для цього слід виконати порівняння декількох моделей, які мають різну архітектуру або різні налаштування параметрів. На цьому кроці буде доречно використати метод перевірки моделі Cross Validation. Він дозволяє оцінити те, як результати статистичного аналізу будуть узагальнюватися до незалежного набору даних. Також в ході навчання потрібно вирішити проблему overfitting та underfitting, якщо вони виникають. Зазвичай в подібних задачах частіше трапляється саме перенавчання, тобто модель показує бездоганний результат на навчальній вибірці і поганий на безпосередньо тестовій. Проте використовуючи підхід cross validation дану проблему можна виявити і вже після цього змінювати налаштування моделі. Окрім цього, необхідно визначити метрики якості прогнозу, які будуть прямо впливати на вибір найкращого класифікатора.

Результатом роботи виступає набір міток хвороб, сформований для нових, нерозмічених зображень. Для безпосередньо перевірки якості даного набору, доцільно використати можливість ресурсу Kaggle в рамках змагання зробити submission своїх результатів. Система містить правильні відповіді та автоматично вираховує відповідність між ними та внесеними даними.

1.5 Висновки до розділу 1

Як бачимо, необхідність розробки систем розпізнавання патологій харчових культур є надзвичайно високою в силу запровадження автоматизації у зв’язку з масштабуванням та нарощуванням потужностей виробництва. Фермери зацікавлені в використанні автономних систем класифікації захворювань насаджень, оскільки це дозволить швидше проводити діагностику. Своєчасне виявлення патологій призведе до оперативного втручання в життєвий цикл рослин, ефективного лікування та допоможе запобігти поширенню. Це в свою чергу покращить обсяг та якість фінальної продукції, та дозволить мінімізувати

втрати пов’язані з необачним ставленням до стану дерев. Також на актуальність даної проблеми вказує факт створення подібного змагання на ресурсі Kaggle. Тобто спеціалісти з усього світу мають змогу вирішити поставлену задачу, оскільки розроблені конкретні підходи можна застосовувати до різних продовольчих культур, що може вплинути на технології вирощування продуктів харчування незалежно від місця розташування та напрямку діяльності виробництва.

Проте, незважаючи на розвиток та наявність широкого спектру можливостей, дослідники також отримують велику кількість різнопланових та різносторонніх проблем, пов’язаних безпосередньо з зображеннями. Дані особливості задач комп’ютерного зору мають бути враховані при розробці та експлуатації систем розпізнавання.

Задача класифікації патологій рослин насправді представляє класичну задачу навчання з учителем, оскільки ми маємо розмічений експертами навчальний набір. У зв’язку з цим можна застосувати велике різноманіття методів машинного навчання. Це можуть бути як класичні методи класифікації так і використання нейронних мереж. Насамперед, основним завданням будь-якої побудованої моделі є якісне прогнозування на нових даних, що дозволить розробленій системі приносити практичну користь та вдосконалити технологічні процеси.

РОЗДІЛ 2
МАТЕМАТИЧНІ ОСНОВИ МЕТОДІВ РОЗПІЗНАВАННЯ

2.1 Класифікатори

В попередньому розділі ми чітко формалізували задачу дослідження. Вхідний набір даних розмічений на окремі категорії захворювань, тому розпізнавання патологій листя рослин зводиться до задачі класифікації, яка є підкласом задач навчання з учителем. Як було сказано раніше, для роботи класифікаторів з зображеннями необхідно для початку сформувати набори характерних ознак кожного екземпляру. Для вирішення цієї підзадачі використовують дескриптори зображень. В даному розділі ми більш детально розглянемо алгоритми машинного навчання, які використовуються для подібних задач класифікації зображень. Серед методів які гарно себе зарекомендували можна виділити наступні: логістична регресія, Random Forest, SVM.

Але варто зазначити, що складність поставленої задачі полягає в наявності більше ніж одної хвороби на деяких екземплярах навчального набору. Це так звана проблема Multi-Label Classification, і в рамках класичних методів її пропонують вирішувати наступним чином: тренуються окремі класифікатори для кожної категорії і потім агрегуються в одну модель. Під час обробки тестовий екземпляр оброблюється кожним з класифікаторів потім отримані категорії об’єднуються в одне передбачення. Саме тому, далі ми розглянемо архітектуру класичних бінарних класифікаторів, які в якості ознак приймають результати роботи дескрипторів зображень.

2.1.1 Логістична регресія

Логістична регресія є одною з найбільш поширених моделей бінарної класифікації. Вона виникла як варіація лінійної регресії, але для задач де

необхідно прогнозувати категоріальну змінну. Замість вибору оптимальної прямої або гіперплощини для опису даних, в логістичній моделі використовується функція, яка дозволяє стискати вхідні дані на проміжок [0;1]. Вона визначається наступним чином(рис. 2.1):

$$logistic(\eta) = \frac{1}{1 + exp\ (-\eta)}$$

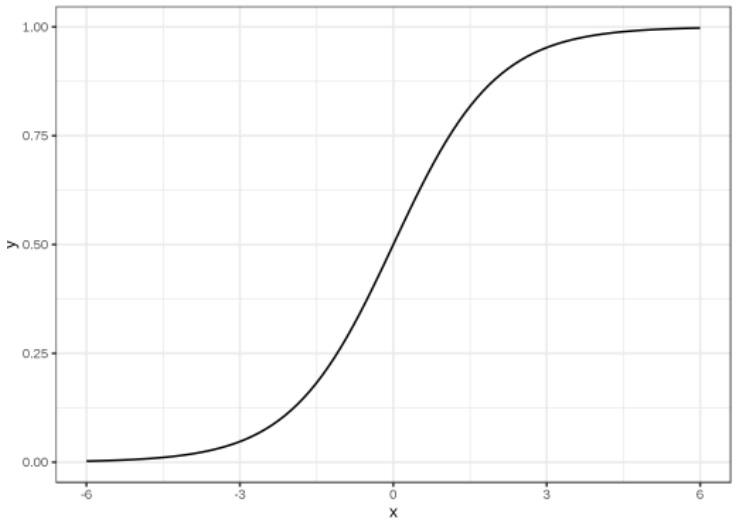


Рисунок 2.1 – Графік логістичної функції

Дана функція прогнозування повертає оцінку ймовірності приналежності класу. Для того, щоб зробити прогноз та сформувати мітку (істина / хибність, кішка / собака), ми вибираємо деяке порогове значення або границю відсікання. Якщо значення ймовірності більше за встановлений поріг – ми відносимо його до першої категорії, якщо нижче - до другої. Вибір порогу відсікання може суттєво вплинути на якість прогнозу, та дозволяє налаштовувати модель під особливості конкретної задачі.

Алгоритм виглядає наступним чином:

1) Функція-гіпотеза

$$h_\theta\left(x^{(i)}\right) = logistic\left(\theta_0 + \theta_1 x_1^{(i)} + \cdots + \theta_n x_n^{(i)}\right)$$

де θ - вектор вагів,

$x^{(i)}$– вектор ознак *i*-того запису в навчальній вибірці,

n- кількість факторів.

2) Функція втрат:

$$J(\theta) = -\frac{1}{m}\sum_{i-1}^{m}[y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) + (1-y^{(i)})\log\left(1-h_\theta\left(x^{(i)}\right)\right]$$

де m – кількість записів в тренувальному наборі

3) Мінімізація функції втрат по θ. Методи які використовуються: Gradient Descent, BFGS та інші градієнтні методи. В результаті отримуємо оптимальні значення вагів.

4) Обчислення ймовірності приналежності одному з класів(0 або 1) для тестових даних:

$$P\left(y^{(i)}=1\right) = \frac{1}{1+exp\left(-\left(\theta_0+\theta_1 x_1^{(i)}+\cdots+\theta_n x_n^{(i)}\right)\right)}$$

$$P\left(y^{(i)}=0\right) = 1-P\left(y^{(i)}=1\right) = 1-\frac{1}{1+exp\left(-\left(\theta_0+\theta_1 x_1^{(i)}+\cdots+\theta_n x_n^{(i)}\right)\right)}$$

5) Спираючись на значення порогу відсікання (зазвичай threshold=0.5) прогнозування відповідного класу:

- якщо $P\left(y^{(i)}=1\right) \geq threshold$, то клас 1;
- якщо $P\left(y^{(i)}=1\right) < threshold$, то клас 0.

Окрім проблем пов’язаних з вибором вдалого порогу відсікання, як і будь-якій регресійній моделі, логісті притаманного перенавчання. Для того, щоб уникати подібних ситуацій можна скористатись регуляризацією. Суть методу полягає в додаванні до Cost Function ще одного доданка, який «штрафує» функцію, якщо підбираються надто великі значення вагів. Серед

найпоширеніших видів: L1 або регуляризація Лассо, L2 або регуляризація Ріджа та ElasticNet- комбінація L1 та L2.

1) L1:

$$\hat{J}(\theta) = J(\theta) + \lambda \sum_{j=1}^{n} |\theta_j|$$

2) L2:

$$\hat{J}(\theta) = J(\theta) + \lambda \sum_{j=1}^{n} \theta_j^2$$

3) ElasticNet:

$$\hat{J}(\theta) = J(\theta) + \lambda_1 \sum_{j=1}^{n} |\theta_j| + \lambda_2 \sum_{j=1}^{n} \theta_j^2$$

де λ, λ_1 , λ_2 – невід'ємні параметри що називається коефіцієнтами регуляризації.

Плюси використання логістичної регресії полягають у простоті реалізації та відсутності великої кількості гіперпараметрів, які необхідно налаштовувати. Також даний метод підраховує безпосередньо ймовірності. Це суттєва перевага перед моделями, які можуть дати лише остаточну класифікацію. Знання того, що екземпляр має 99% ймовірність приналежності до класу порівняно з 51%, має велике значення. Логістична регресія вимагає помірної або відсутності мультиколінеарності між незалежними змінними. Це означає, що якщо дві незалежні змінні мають високу кореляцію, слід використовувати лише одну з них. Повторення інформації може призвести до неправильної оцінки ваг під час

мінімізації функції витрат. Також дана модель досить чутлива до наявності викидів в даних.

2.1.2 Random Forest

Випадковий ліс - це один з алгоритмів навчання з учителем.. «Ліс» який він будує являє собою ансамбль дерев рішень, який зазвичай навчають методом bagging. Термін bagging є скороченням від bootstrap aggregation і головна ідея цього рішення полягає в тому, що поєднання низки моделей навчання підвищує загальний результат. Для random forest це працює наступним чином:

- загальна множина факторів розбивається на підгрупи та для кожної з них будується своє просте дерево прийняття рішень, можливо не найбільш вдале, але подібний підхід дозволяє зменшити кореляцію між навченими моделями;
- результати роботи всіх навчених дерев агрегуються шляхом голосування щоб остаточно сформувати значення класу.

Дерево рішень - будівельний матеріал для випадкового лісу, є інтуїтивно зрозумілою моделлю. Ми можемо сприймати дерево прийняття рішень як низку запитань так / ні щодо наших даних, які в кінцевому підсумку ведуть до передбачуваного класу. Дану модель можна представити у вигляді графу, де ребра відповідають за розгалуження так/ні, а вузли містять інформацію про приналежність класу в залежності від такого розгалуження. Також в вузлах(листках) зберігається значення ознак екземпляру та значення інформаційного критерію, в залежності від якого відбувається розгалуження дерева. Приклад структури дерева рішень показаний на рисунку 2.2.

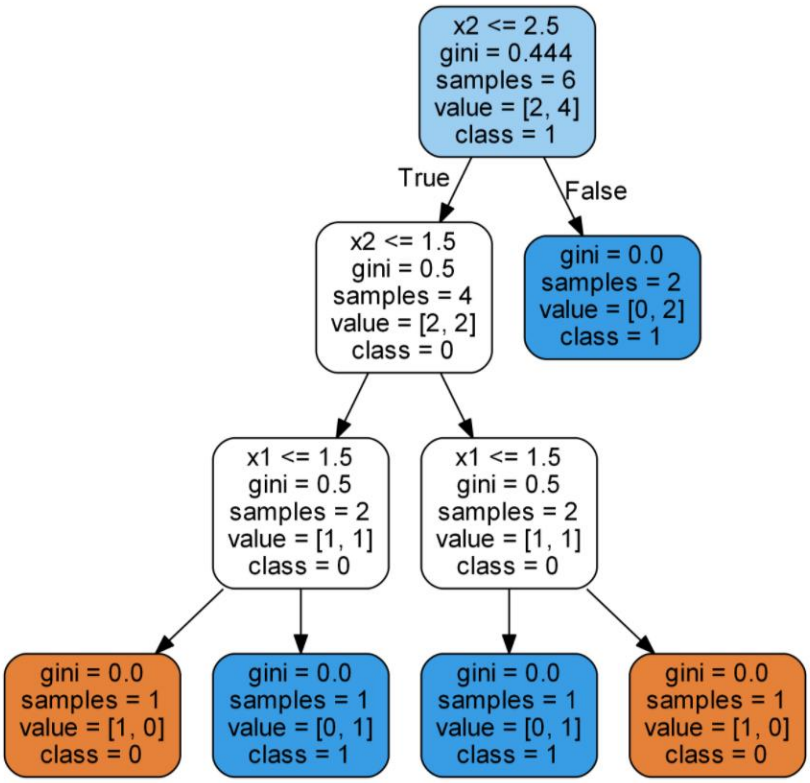


Рисунок 2.2 – Приклад дерева рішень для бінарної класифікації об’єкта з двома ознакам

Технічні деталі дерева рішень полягають у тому, як формуються питання щодо даних. В алгоритмі CART(Classification and Regression Tree) дерево рішень будується шляхом визначення розбиттів, які при відповіді призводять до найбільшого зменшення критерію Джині. Це означає, що дерево рішень намагається сформувати вузли, що містять велику частку зразків з одного класу, знаходячи значення ознак в об’єктах, які чітко ділять дані на класи. Значення критерію Джині у вузлі - це ймовірність неправильної класифікації нового екземпляру даних, якщо цей новий екземпляр був випадково класифікований відповідно до розподілу ймовірностей членства класу в межах кожної підмножини. Так значення критерію в вузлі *n* наступним чином:

$$I_G(n) = 1 - \sum_{i=1}^{J} (p_i)^2$$

де J-кількість класів,

p_i- ймовірність приналежності екземпляру i-тому класу.

На кожному вузлі дерево рішень здійснює пошук функцій для розділення значення, що призводить до найбільшого зменшення значення критерію розгалуження. (Альтернативою критерію Джині може виступати коефіцієнт посилення, інформаційна відстань). Потім він повторює цей процес розщеплення в жадібній, рекурсивній процедурі, поки не досягне максимальної глибини, або кожен вузол буде містити лише зразки одного класу. Повна зважена загальна оцінка Джині(зважується часткою точок від батьківського вузла в цьому вузлі) на кожному рівні дерева повинна зменшуватися. Зрештою, зважене загальне значення критерію останнього шару дорівнює 0, тобто кожен вузол є абсолютно однозначним, і немає шансів, що випадково обрана з цього вузла точка буде неправильно класифікована. Хоча це може здатися позитивним, це означає, що модель може потенційно перенавчатись, оскільки вузли будуються лише з використанням навчальних даних.

Причина, по якій дерево рішень схильне overfitting, у випадку коли ми не обмежуємо максимальну глибину, полягає в тому, що воно має необмежену гнучкість, а це означає, що воно може продовжувати рости, поки не матиме рівно одного листового вузла для кожного окремого спостереження, досконало класифікуючи їх усі. Якщо повернутися до зображення дерева рішень і обмежити максимальну глибину до 2 (роблячи лише один поділ), класифікації перестали бути правильними на 100%. Ми зменшили дисперсію дерева рішень, але ціною збільшення зсуву(bias). Як альтернативу обмеженню глибини дерева, яке зменшує дисперсію (добре) та збільшує зсув (погано), ми можемо об’єднати багато дерев рішень в єдину модель ансамблю, відому як випадковий ліс. Тобто багато не дуже якісних невеликих дерев вкупі можуть показувати гарні результати.

Серед переваг використання такого методу класифікації можна виділити стійкість до перенавчання, якщо ви підбираєте достатню кількість дерев. Також навіть з обраними за замовчуванням гіперпараметрами модель показує гарні

результати. Основне обмеження випадкового лісу полягає в тому, що велика кількість дерев може зробити алгоритм занадто повільним та неефективним для прогнозів у реальному часі. Як правило, ці алгоритми швидко тренуються, але досить повільно створюють прогнози після того, як їх навчать. Для більш точного прогнозування потрібно більше дерев, що призводить до уповільнення моделі. У більшості реальних додатків алгоритм випадкового лісу є досить швидким, але, безумовно, можуть бути ситуації, коли продуктивність виконання є важливою, і інші підходи будуть кращими.

2.1.3 Support Vector Classification

Ще одним потужним методом класифікації є метод опорних векторів або SVC який належить до підрозділу Support Vector Machine(SVM). Як і попередні розглянуті методи він дозволяє вирішувати поставлену задачу розпізнавання приналежності листків до окремих категорій хвороб, використовуючи значення ознак після роботи дескрипторів. SVC відрізняється від інших алгоритмів класифікації через те, як він обирає межу прийняття рішення, яка максимізує відстань від найближчих точок даних усіх класів. Межа прийняття рішення, створена SVC, називається класифікатором максимального поля або гіперплощиною максимального поля. Далі розглянемо принцип роботи методу опорних векторів в двовимірному просторі, тобто коли кількість ознак дорівнює 2. В такому випадку кожне спостереження з навчально набору буде лише точкою на площині.

Задача простого лінійного класифікатора SVM – вибрати пряму яка буде розділяти представників окремих класів. Це означає, що всі екземпляри даних на одній стороні лінії будуть відноситись до однієї категорії, а представники даних на іншій стороні лінії будуть віднесені до іншої категорії. Звідси випливає, що можна обрати нескінченну кількість прямих. Тобто Ви намагаєтесь розділити точки на певні класи, яким вони повинні належати, але ви не хочете, щоб у вас були точки які потрапили не в ту категорію. Отже, лінія розмежування має

знаходитись між двома найближчими точками різних класів і при цьому тримати дистанцію між ними. Для цього дві найближчі точки даних надають Вам опорні вектори, за якими можна знайти межу рішення(рис. 2.3).

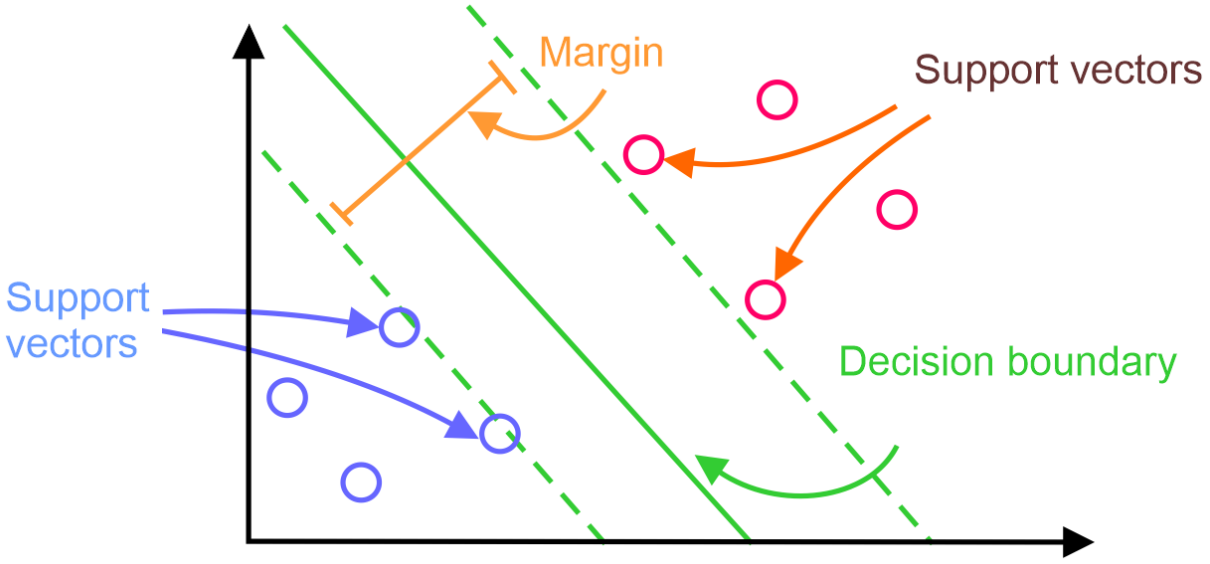


Рисунок 2.3 – Принцип роботи лінійного SVC

Алгоритм роботи виглядає наступним чином:

1) З екземплярів даних у вигляді $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$ обираємо $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \dots, l^{(m)} = x^{(m)}$.
2) Для кожного $x^{(i)}$ обраховуються функції:

$$f_1^{(i)} = similarity(x^{(i)}, l^{(1)})$$
$$f_2^{(i)} = similarity(x^{(i)}, l^{(2)})$$
$$\vdots$$
$$f_m^{(i)} = similarity(x^{(i)}, l^{(m)})$$

3) Формується вектор $f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}$, $f_0 = 1$.
4) Мінімізується функція витрат за умови, що $\theta^{\mathrm{T}} f \geq 0$:

$$\min_{\theta} C \sum_{i=1}^{m} y^{(i)} cost_1\left(\theta^{\mathrm{T}} f^{(i)}\right) - (1 - y^{(i)}) cost_0\left(\theta^{\mathrm{T}} f^{(i)}\right) + \frac{1}{2} \sum_{j=1}^{m} \theta_j^2$$

де $cost_1$ та $cost_0$функції виду(рис.2.4):

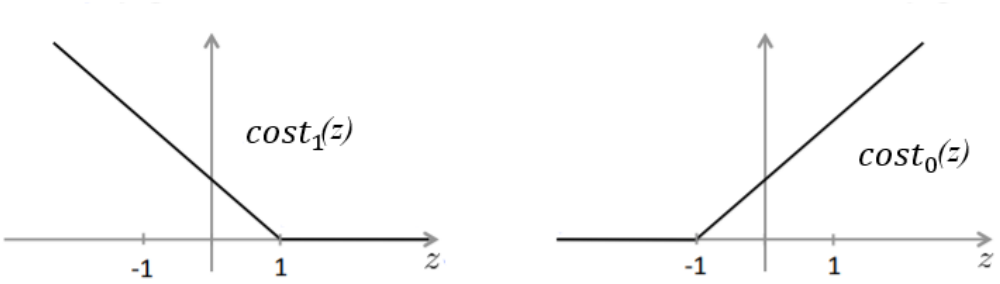


Рисунок 2.4 – Графіки функцій $cost_1$ та $cost_0$

5) Якщо в результаті для $x^{(i)}$: $\theta^{\mathrm{T}} f^{(i)} \geq 0$ то $y^{(i)} = 1$. Якщо $\theta^{\mathrm{T}} f^{(i)} < 0$ то $y^{(i)} = 0$.

У якості функції similarity обирають так зване ядро SVM. Воно може бути лінійним, поліноміальним, сигмоїдом, тощо. Узагалі кажучи, точки не зобов'язані знаходитись чітко по обидві сторони прямої лінії. В такому випадку використовують нелінійне ядро, найчастіше це гаусівська радіальна базова функція(Gaussian Radial Basis Function). RBF визначається так:

$$f(x_1, x_2) = exp\left(-\gamma \|x_1 - x_2\|^2\right)$$

В цій формулі γ визначає скільки одна тренувальна точка має інших точок навколо себе. Приклад нелінійного класифікатора показано на рисунку 2.5.

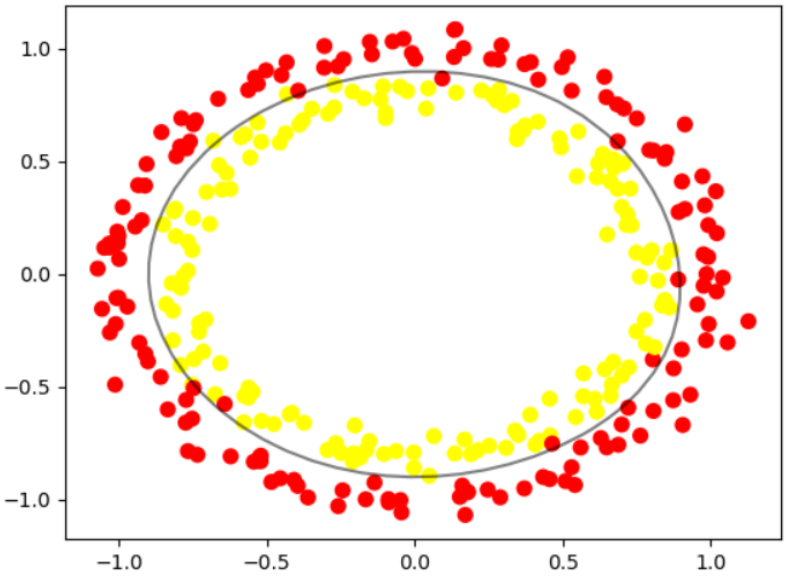


Рисунок 2.5 – Приклад нелінійного SVC з використанням RBF

Авжеж метод не обмежується двовимірним простором і здатен працювати з даними, які мають багато ознак. В такому випадку алгоритм будує деяку гіперплощину, яка відсікає представників різних класів один від одного. Це доволі складно зрозуміти, особливо маючи десятки ознак і багатовимірний простір. Тому часто SVC називають black box алгоритмом.

Що робить лінійний алгоритм SVM кращим, ніж деякі інші алгоритми, наприклад k-найближчих сусідів, це те, що він вибирає найкращу лінію відсікання для класифікації точок даних. Гіперплощина знаходиться якомога далі від точок різних класів, що дозволяє більш надійно локалізувати окремі категорії даних. Проте SVC не надають ймовірнісної оцінки і на великих наборах процес навчання може тривати досить довго.

2.2 Згорткові нейронні мережі

Як вже було сказано раніше, Convolutional Neural Networks є одним з основних методів розпізнавання образів. Дані моделі належать до класу глибинних методів навчання. Їхню перевагу перед іншими нейронними мережами можна пояснити наявністю саме операції згортки Кількість

параметрів у нейронній мережі швидко зростає зі збільшенням кількості шарів. Це може зробити навчання моделі обчислювально важким, а іноді і неможливим. Налаштування такої кількості параметрів може бути дуже непосильним завданням. Але час, необхідний для налаштування цих параметрів, зменшується завдяки згортці. CNN є повнозв’язними нейронними мережами прямого передавання. Вони дуже ефективні у зменшенні кількості параметрів, не втрачаючи при цьому якості моделей. Оскільки безпосередньо такі об’єкти як зображення мають високу розмірність (кожен піксель розглядається як окрема ознака екземпляру), то метод який буде оптимізувати кількість характеристик, залишаючи найбільш значимі, або компонуючи їх в групи, матиме змогу швидше навчатись. Згорткові нейронні мережі відповідають цим вимогам. Крім того, CNN мають змогу визначати краї об'єктів на будь-якому зображенні. Також їхня структура, яка є різновидом багатошарового перцептрона, дозволяє знизити обсяг попередньої обробки зображень у вибірці. У цьому розділі описано детальний принцип роботи згорткових мереж та розглянуто архітектуру найбільш використовуваних моделей. Серед них: VGG, ResNet, DenseNet, EfficientNet.

2.2.1 Загальний принцип роботи CNN

Архітектура ConvNet аналогічна структурі зв’язку нейронів в мозку людини і натхненна організацією зорової кори. Окремі нейрони реагують на подразники лише в обмеженій області зорового поля, відомому як рецептивне поле. Колекція таких полів накладається одна на одну, щоб охопити всю зорову зону. В першому розділі ми з’ясували, що згорткові мережі складаються з: згорткових шарів, функцій активації, пулінг-шарів та повнозв’язного шару. Далі ми розглянемо як працюють ці складові та деякі методи оптимізації роботи.

Процес починається зі згорткового шару. Він представляє собою набір карт ознак(матрицю) у яких є скануюче ядро. Кількість карт визначається вимогами до задачі, але якщо взяти великий обсяг карт, якість розпізнавання збільшиться,

проте також зросте обчислювальна складність. Найчастіше береться відношення 1:2, тобто одна карта попереднього шару пов’язана з двома картами згорткового шару. Розмір (w, h) у всіх карт одного шару однаковий і вираховується за наступною формулою:

$$(w, h) = (mW - kW + 1, mH - kH + 1)$$

де mH – висота попередньої карти,

mW – ширина попередньої карти,

kH – висота ядра,

kW – ширина ядра.

Ядро є так званим фільтром або вікном, яке ковзає по всій області попередньої карти(в першому шарі – вхідне зображення) і знаходить визначені характеристики об’єктів. Наприклад, в задачах розпізнавання обличь, в процесі навчання одне з ядер могло б видавати найбільший сигнал в області очей, носу, роту, інше ядро могло б виявляти інші ознаки. Ядро представляє собою систему окремих синапсів. В звичайній багатошаровій мережі дуже багато зв’язків між нейронами і це суттєво вповільнює процес знаходження ознак. В згортковій мережі все навпаки, загальні ваги дозволяють скоротити число зв’язків і дати змогу знаходити ту саму ознаку по всій області зображення. Розмір ядра зазвичай в границях від 3×3 до 7×7. Якщо розмір ядра занадто малий, то воно не зможе виділити якісь ознаки, якщо надто велике – збільшується кількість зв’язків між нейронами. Також розмір обирається таким чином, щоб розмір карт згорткового шару був парним, це запобігає втраті інформації при зменшенні розмірності в підвибірковому шарі.

На початку значення кожної матриці згорткового шару дорівнюють 0. Значення вагів ядер задаються випадковим чином на проміжку [-0.5;0.5]. Ядро g проходить всю попередню карту з певним кроком(зазвичай 1), значення фільтра перемножуються поелементно з вмістом вікна, і результат записується в

результуючу матрицю(рис. 2.6). Дана операція показує в яких частинах зображення знаходяться ознаки описані фільтром.

$$(f * g)[m,n] = \sum_{k,l} f[m-k, n-l] * g[k,l]$$

де f – вихідна матриця зображення,

g – ядро згортки.

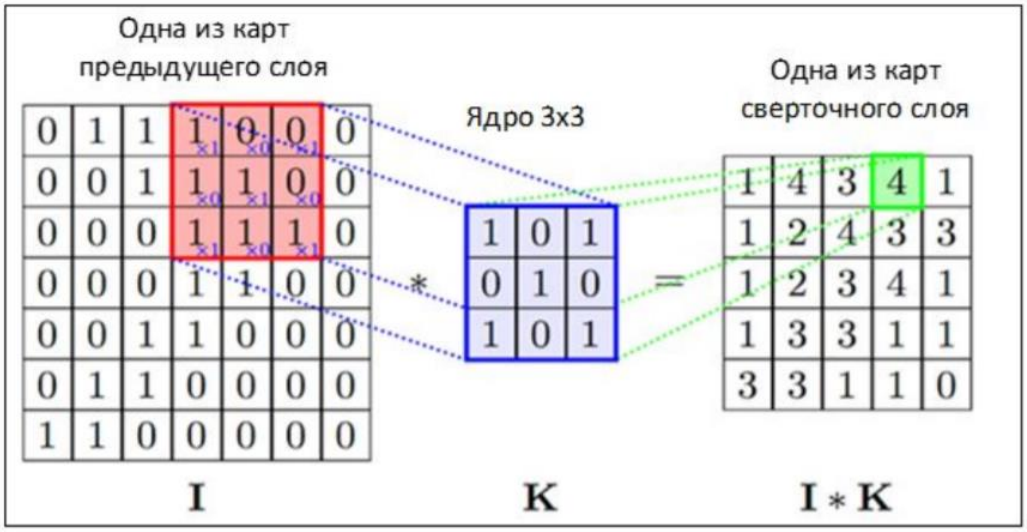


Рисунок 2.6 – Згортка попереднього шару мережі

Функції активації дозволяють зробити мережу нелінійною, оскільки якщо їх не використовувати, або використовувати лінійну функцію, то будь-яку кількість шарів мережі можна було б замінити одним шаром. Відомо, що нейронні мережі здатні наблизити скільки завгодно складну функцію, якщо в них присутня достатня кількість шарів і активатор є нелінійним. Але поширеною проблемою використання нелінійних функцій є затухання та збільшення градієнтів. Це викликає недонавчання та перенавчання відповідно під час backpropagation. Можна виділити такі основні функції активації, які використовують в згорткових мережах:

- сигмоїда

$$\sigma(x) = \frac{1}{1+e^{-x}};$$

- гіперболічний тангенс

$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}};$$

- ReLU(Rectified Linear Unit)

$$R(x) = max\ (0, x).$$

Сигмоїда є неперервною, монотонно зростаючою функцією яка завжди приймає значення на проміжку [0;1]. В процесі оберненого поширення помилки локальний градієнт множиться на загальний, тому якщо локальний градієнт досить малий, як у випадку сигмоїдної функції, то він фактично обнуляє загальний. В результаті сигнал майже не буде проходити через нейрон. Також варто бути обережними під час ініціалізації сигмоїдних нейронів, тому що, якщо початкові ваги мають надто великі значення, більшість нейронів одразу стане насиченими і модель буде погано навчатись.

Гіперболічний тангенс має область визначення [-1;1], а, отже, є симетричним відносно 0. Це дозволяє градієнту рухатись в різних напрямках. Проте як і у випадку сигмоїди, тангенсу притаманне затухання чи збільшення градієнта. Слід сказати, що ці дві функції є також дуже ресурсомісткими , оскільки потребують таких операцій як піднесення до степеня. Тому в задачах класифікації зображень разом зі згорткою використовують функцію ReLU.

Плюсом використання ReLU є те, що її похідна дорівнює або 0, або 1 і тому не може виникнути затухання або збільшення градієнту. Також вона є менш ресурсозатратною в порівнянні з tanh та сигмоїдою. Вона дозволяє відсікати непотрібні деталі та проріджувати ваги. Але даний активатор також має свої недоліки. Наприклад, великий градієнт, що проходить через ReLU, може

призвести до оновлення вагів таким чином, що даний нейрон ніколи більше не активується. Це може траплятись коли швидкість навчання є занадто великою. Але не зважаючи на це, саме ReLu найчастіше використовують в задачах розпізнавання образів. Приклад роботи ReLu показаний на рисунку 2.7.

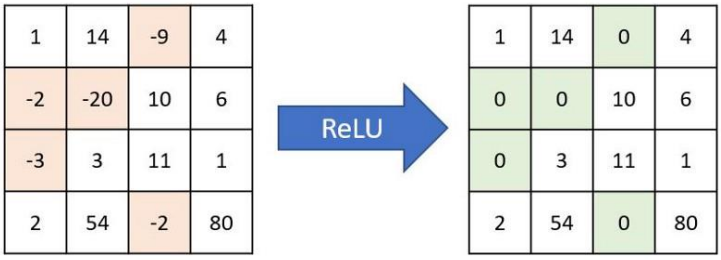


Рисунок 2.7 – Приклад роботи ReLU

Зазвичай наступний шар після згорткового – це пулінг-шар. Він дозволяє стиснути інформацію про знайдені характеристики, шляхом відкидання ознак низького рівня. Фільтри в згортковому шарі записують інформацію для кожного пікселя, але вікно зазвичай захоплює сусідні пікселі, тому трапляються повтори інформації про одні й ті самі області. Існують два типи пулінгу: max-pooling і average-pooling. Суть підходів доволі проста за виключенням того що один метод вираховує максимальне значення ознаки в області, а інший – середнє. Частіше використовують max-pooling, тому що виділення середніх значень для подальшої роботи може не виділити всі якісні ознаки. Розмір вікна пулінгу зазвичай 2×2. Вікно пересувається вздовж вхідного зображення обчислюючи максимальне значення та записуючи його в нову матрицю ознак. Таким чином отримуємо зменшене зображення, з відібраними максимальними пікселями, які найкращим чином описують область(рис.2.8).

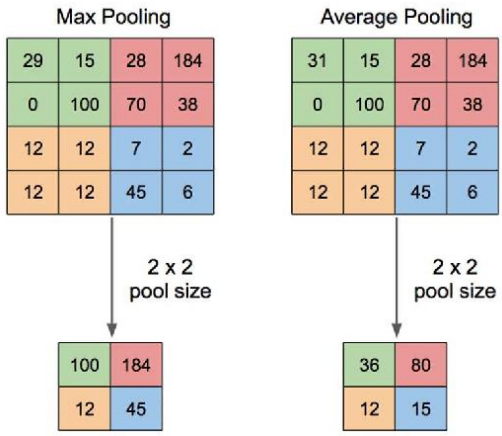


Рисунок 2.8 – Приклад роботи пулінг-шару

В результаті чередування шарів згортки та пулінгу, кількість яких може бути довільна, ми отримуємо розбиття початкового зображення на велику кількість окремих якісних характерних ознак. На цьому етапі застосовують згладжування. Ви перетворюєте об’єднану карту ознак у послідовний стовпець чисел (довгий вектор). Це дозволяє цій інформації стати вхідним шаром штучної нейронної мережі для подальшої обробки. Далі йде повноз’язний шар, який на основі поданих ознак приймає рішення до якого класу належить екземпляр. Повнозв'язний шар - це традиційний багатошаровий персептрон. Основною метою штучної нейронної мережі є поєднання наших ознак у більшу кількість атрибутів класу. Процес прийняття рішення проходить наступним чином: матриця вагів W множиться на вектор ознак x_i , додається величина зсуву b, звідси отримуємо оцінку для кожного класу(рис. 2.9).

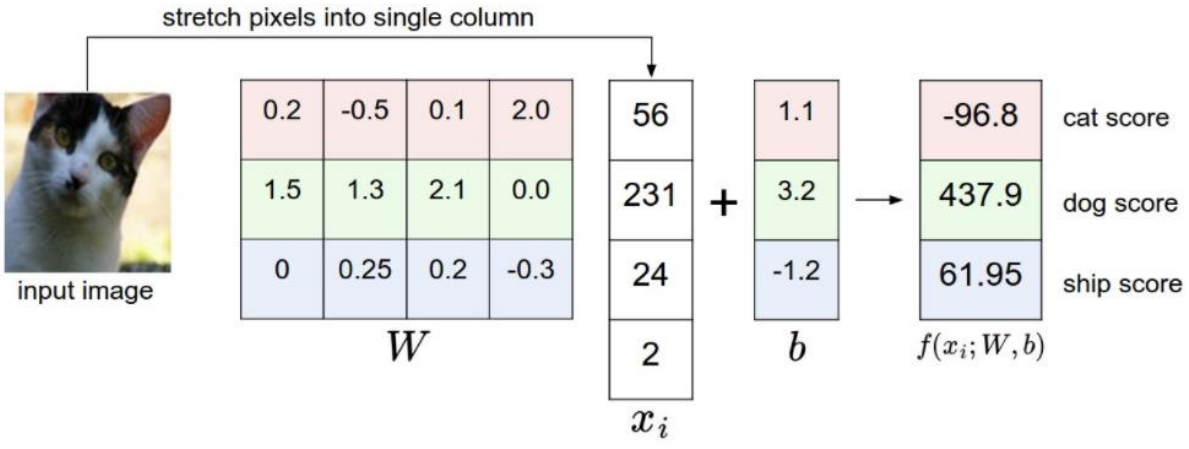


Рисунок 2.9 – Класифікація в повнозв’язному шарі

Отримані результати зазвичай подають в функцію активації softmax для визначення ймовірності приналежності кожному з класів. Активатор переводить значення оцінки на проміжок [0;1], яке можна інтерпретувати як ймовірність приналежності класу. Softmax*:*

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

де z_j – значення оцінки *j*-го класу,

K – кількість класів.

Далі обчислюється функція втрат, а потім виконується алгоритм зворотнього поширення помилки. Ваги та фільтри згортки налаштовуються для мінімізації функції втрат. За функцію витрат найчастіше беруть Binary Cross-Entropy Loss.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i log\ (p(y_i)) + (1 - y_i) log\ (1 - p(y_i))$$

де y_i – істине значення класу(1 або 0) для *i*-го зразка,

$p(y_i)$ – ймовірність, того що зразок належить класу 1.

Використання перехресної ентропії допомагає мережі оцінити навіть крихітну помилку і швидше дійти до оптимального стану.

Метод backpropagation умовно можна розділити на 4 блоки: пряме поширення, оцінку функції втрат, зворотнє поширення та оновлення матриці вагів і ядра згортки. В першому навчальному прикладі, значення фільтру та вагів

обирається випадковим чином. Під час прямого поширення береться навчальний зразок , який проходить через всі шари мережі, виокремлює особливі ознаки зображення, далі підраховується ймовірність приналежності певному класу. Обчислюється значення функції втрат. Нашою задачею є мінімізація втрат, тому нам потрібно визначити які значення в матриці вагів призводять до втрат та помилок мережі. Якщо представити візуально, то втрати це трьохвимірний графік, а ваги це незалежні змінні(звісно їх більше ніж 2, але для наочності спростимо). Нам необхідно зменшити втрати, шляхом спуску до точки мінімуму графіка(рис. 2.10). Для цього потрібно знайти похідну функції втрат від значення вагів.

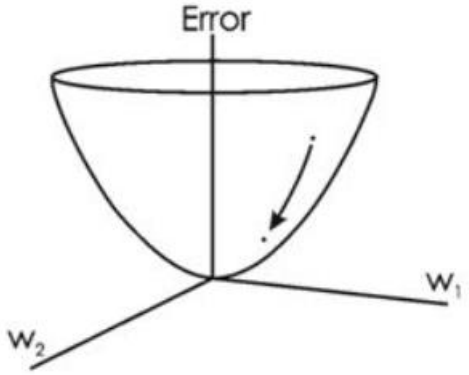


Рисунок 2.10 – Схема роботи зворотнього поширення

Серед методів які застосовуються для оптимізації функції витрат в згорткових нейронних мережах можна виділити наступні: градієнтний спуск, стохастичний градієнтний спуск, методи Adam та Rectified Adam. Після обчислення похідної необхідно оновити значення в матриці вагів:

$$w = w_i - \eta \frac{dL}{dW}$$

де w_i – попередні значення,

L - функція втрат,

η - швидкість навчання.

Швидкість навчання – це гіперпараметр, значення якого налаштовується розробником. Велика швидкість навчання означає, що для оновлення вагів робляться більші кроки, тому зразку може знадобитись менше часу, щоб набрати оптимальну матрицю вагів. Але занадто велика швидкість може призвести до недостатньо точних кроків, які будуть заважати досягненню оптимальних показників. Час за який один екземпляр навчального набору проходить всі етапи, називають періодом дискретизації або епохою. В теорії, коли останній тренувальний зразок закінчить оновлення параметрів, мережа повинна бути достатньо гарно навчена і ваги шарів мають бути знайдені правильно. Далі ми розглянемо архітектуру відомих згорткових мереж, які використовуються в задачах комп'ютерного зору.

2.2.2 VGG

VGG16 - це модель згорткової нейронної мережі, запропонована K. Simonyan та A. Zisserman з Оксфордського університету в роботі «Very Deep Convolutional Networks for Large-Scale Image Recognition». Модель досягає 92,7% точності для top-5 в ImageNet, що є набором даних із понад 14 мільйонів зображень, що належать до 1000 класів. Це була одна з відомих моделей, поданих на ILSVRC-2014(ImageNet Large-Scale Visual Recognition Challenge). Вона працює краще в порівнянні з AlexNet, замінюючи фільтри з великим розміром ядра (11 і 5 у першому та другому згортковому шарі відповідно) на кілька фільтрів розміром 3 × 3 один за одним. VGG16 навчалась протягом кількох тижнів і використовувала графічний процесор NVIDIA Titan Black. Архітектура показана на рисунку 2.11.

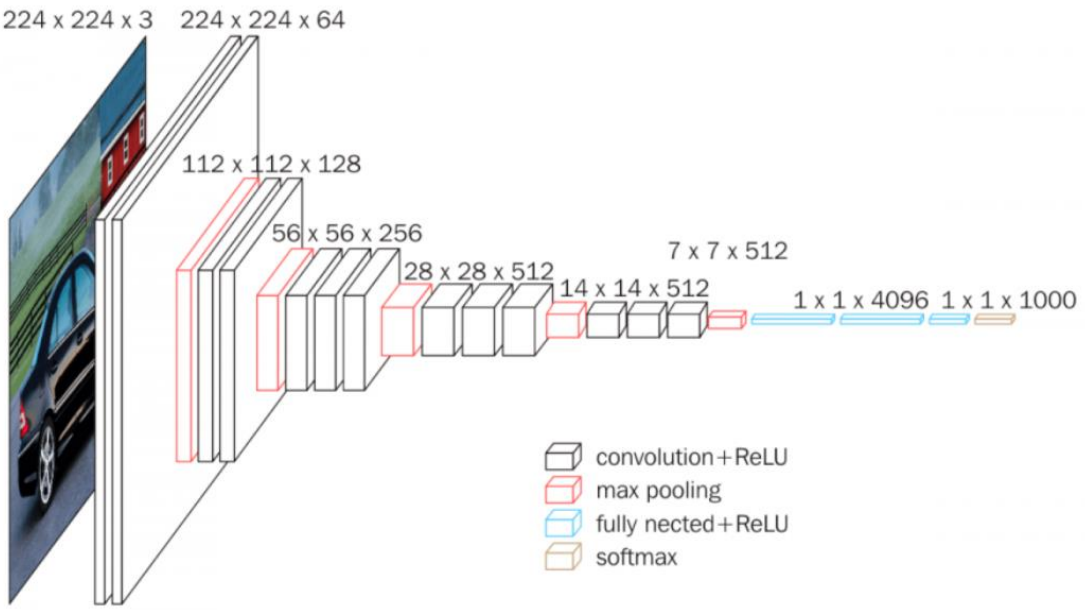


Рисунок 2.11 – Архітектура мережі VGG16

Дані вхідного шару мають фіксований розмір зображення 224 x 224 пікселі та простір кольору RGB. Зображення передається через послідовність згорткових шарів, фільтри яких були розміром 3 × 3, а це в свою чергу найменший розмір для розуміння напрямку ліворуч / праворуч, вгору / вниз, центр). В одній з конфігурацій мережі також використовуються фільтри 1 × 1, які по суті є просто лінійними перетвореннями вхідних карт. Але всі згорткові шари містять нелінійну функцію-активації ReLU. Крок згортки дорівнює 1 пікселю. Пулінг всіх трьох каналів простору RGB здійснюється за допомогою п'яти max-pooling шарів, які почергово слідують за деякими згортковими шарами. Вікно пулінгу розміром 2 × 2, та не перетинається. Три повністю з'єднаних (FC) шари слідують після набору згорткових шарів. Перші два мають по 4096 каналів, третій виконує класифікацію на 1000 (по одному на кожен клас з датасету ImageNet). Кінцевим шаром є активатор softmax.

Перевагами даної мережі є велика кількість конфігурацій та шарів, що дозволяють краще навчити модель. Але VGG занадто повільно працює і потребую великої обчислювальної потужності, у зв’язку з наявністю 3 великих повнозв’язних шарів.

2.2.3 ResNet

ResNet, скорочення від Residual Network - це специфічний тип нейронної мережі, який був представлений в 2015 році Kaiming He, Xiangyu Zhang, Shaoqing Ren та Jian Suny своїй роботі "Deep Residual Learning for Image Recognition". Модель зайняла 1-е місце у класифікаційному змаганні ILSVRC 2015 з коефіцієнтом помилок 3,57% для top-5. Вона базується на методі додавання блоків залишків між згортковими шарами. Зазвичай для того, щоб вирішити складну проблему, ми додаємо кілька нових шарів у глибоких нейронних мережах, що призводить до підвищення точності та продуктивності. Інтуїція додавання нових шарів полягає в тому, що ці шари поступово вивчають більш складні ознаки. Наприклад, у разі розпізнавання зображень, перший шар може навчитися виявляти краї, другий шар може навчитися ідентифікувати текстури і, аналогічно, третій шар може навчитися виявляти об'єкти, тощо. Але було також виявлено, що існує максимальний поріг глибини для згорткової нейронної мережі. Використання residual blocks дозволяє уникнути проблему затухання градієнта при поширенні вглиб.

По-перше існує зв’язок, який пропускає деякі шари мережі(можуть відрізнятися в різних моделях). Це з'єднання називається «skip connection» і є ядром залишкових блоків. Через наявність такого зв’язку, вихідна карта шару буде не однаковою. Так, без використання skip connection, вхідне значення x помножується на ваги шару з подальшим додаванням зміщення. Далі цей вираз проходить через функцію активації f і ми отримуємо результат як $H(x)$. $H(x) = f(wx + b)$ або $H(x) = f(x)$. Але з використання блоків залишку результат змінено на $H(x) = f(x) + x$. У випадку, коли розмірність $f(x)$ відрізняється від x, ми можемо взяти два підходи: skip connection заповнено зайвими нульовими записами, щоб збільшити його розміри; використовується метод проекції, що

робиться шляхом додавання 1 × 1 згорткових шарів до вхідного. У такому випадку виходить: *H (x) = f (x) + w1x.* Схема роботи показана на рисунку 2.12.

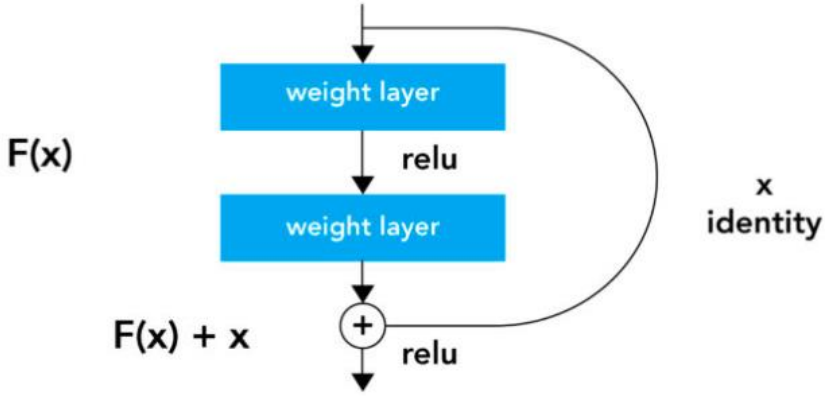


Рисунок 2.12 – Схема роботи residual block

Перевага додавання цього типу з'єднання швидкого доступу полягає в тому, що якщо будь-який шар погіршує продуктивність архітектури, він буде пропущений шляхом регуляризації. Отже, це призводить до навчання дуже глибокої нейронної мережі без проблем, спричинених затуханням або вибухом градієнта. На рисунку 2.13 показано архітектуру мережі ResNet-34.

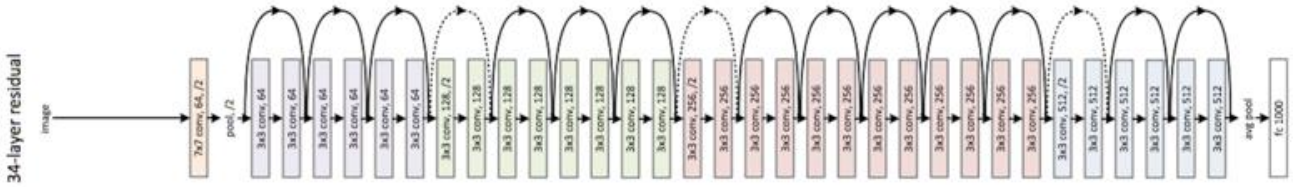


Рисунок 2.13 – Архітектура ResNet з використанням 34 блоків-залишків

Використання ResNet гарантує, що більш глибока мережа ж блоками-залишків буде працювати не гірше за неглибоку, в більшості випадків вона працює краще.

2.2.4 DenseNet

DenseNet (Densely Connected Convolutional Network) була запропонована в 2017 році. Успіх ResNet дозволив припустити, що коротке з'єднання в CNN

дозволяє навчати більш глибокі та точні моделі. Автори проаналізували це спостереження і представили компактно з'єднаний(dense) блок, який поєдную кожен шар з будь-яким іншим шаром. Також, на відміну від ResNet, ознаки перед тим як бути переданими в наступний шар не додаються, а конкатенуються(channel-wise concatenation) в єдиний тензор. Тобто замість $x_l = H_l(x_{l-1}) + x_{l-1}$, використовується $x_l = H_l([x_0, x_1, \dots, x_{l-1}])$. Вихідний сигнал попереднього шару виконує роль вхідного сигналу для другого шару за допомогою операції складеної функції. Ця складена операція складається із рівня згортки, пулінг-шару, batch-нормалізації та нелінійної функції активації. Це означає, що мережа має *L (L + 1) / 2* прямих з'єднань, де *L* - кількість шарів в архітектурі. Приклад архітектури показано на рисунку 2.14.

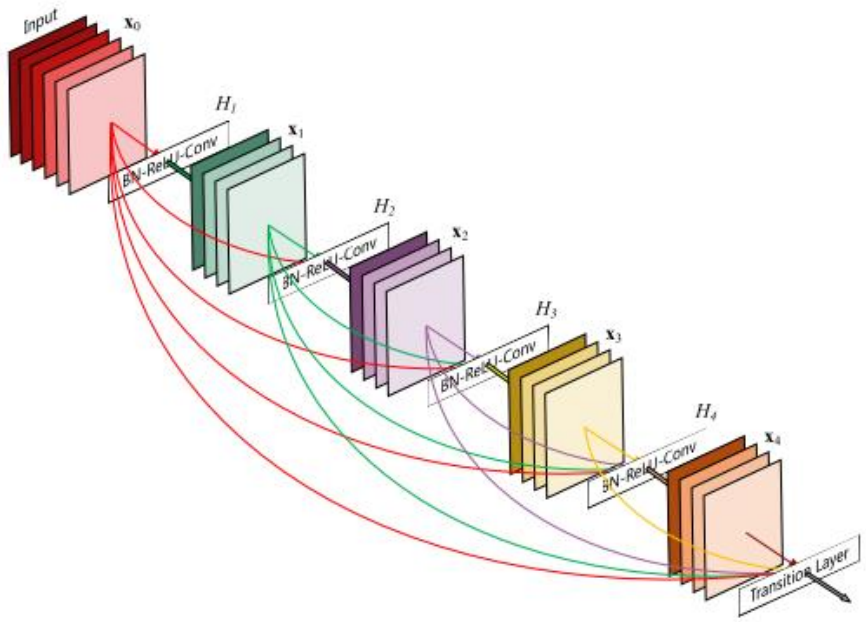


Рисунок 2.14 – Архітектура DenseNet з використанням 5 dense-блоків

Оскільки кожен шар отримує всі попередні шари як вхідні, це дозволяє отримати більш різноманітні ознаки та більш багаті шаблони. При цьому кількість параметрів мережі DenseNet набагато менша, ніж у моделей з такою самою точністю роботи. Це дозволяє використовувати менше часу та ресурсів процесора. Також, завдяки своїй структурі, модель стійка до затухання або збільшення градієнта.

2.2.5 EfficientNet

«EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks» була представлена в 2019 році і мала на меті боротьбу з масштабуванням мереж. Існує три основні розмірності CNN: глибина, ширина та resolution. Глибина просто означає, наскільки глибокими є мережі, що еквівалентно кількості шарів у ній. Ширина просто означає, наскільки широка мережа. Наприклад, однією мірою ширини є кількість каналів кольору у згортковому рівні. Тоді як resolution - це просто роздільна здатність зображення, яке передається в CNN.

Масштабування мережі за глибиною - це найпоширеніший спосіб масштабування. Глибину можна як збільшити, так і зменшити, додаючи/видаляючи шари відповідно. Наприклад, ResNets можна масштабувати з ResNet-50 до ResNet-200, а також можна масштабувати з ResNet-50 до ResNet-18, якщо дозволяють технічні можливості. Але навіщо масштабувати глибину? Інтуїція полягає в тому, що більш глибока мережа може захоплювати багатші та складніші функції та добре узагальнює нові завдання. Проте чим більша кількість шарів, тим більша ймовірність затухання градієнта. Навіть якщо ви уникаєте зменшення градієнтів, а також використовуєте деякі прийоми, щоб зробити тренування плавним, додавання більшої кількості шарів не завжди допомагає. Наприклад, ResNet-1000 має таку саму точність, що і ResNet-101. Масштабування в глибину зазвичай використовується, коли ми хочемо зберегти свою модель невеликою. Ширші мережі, як правило, можуть захоплювати більш дрібні деталі. Крім того, менші моделі легше тренувати. Але при різкому збільшенні ширини, ріст збільшення точності буде спадати. Така ж сама ситуація і з масштабуванням роздільної здатності. Наприклад, збільшення роздільної здатності вхідних зображень з 500×500 до 560×560 не дає значних покращень. Тому запропоновано підхід в якому здійснюється масштабування за всіма трьома параметрами. У ході дослідження автори встановили що ефективно на ріст

точності впливає безпосередньо одночасне зростання розмірностей мережі. Але також важливо збалансувати всі параметри для покращення якості. Автори запропонували просту, але дуже ефективну техніку масштабування, яка використовує складений коефіцієнт ϕ для рівномірного масштабування ширини, глибини та роздільної здатності мережі наступним чином(рис. 2.15):

$$\begin{aligned} \text{depth: } & d = \alpha^\phi \\ \text{width: } & w = \beta^\phi \\ \text{resolution: } & r = \gamma^\phi \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned}$$

Рисунок 2.15 – Схема масштабування розмірностей в EficcientNet

У CNN шари згортки є найбільш обчислюваною частиною мережі. Крім того, FLOPS(кількість операцій за секунду, міра швидкодії) регулярної згортки майже пропорційний d, w^2, r^2, тобто подвоєння глибини подвоїть FLOPS, тоді як подвоєння ширини або роздільної здатності збільшує FLOPS майже в чотири рази. Отже, для того, щоб переконатись, що загальна кількість FLOPS не перевищує 2^ϕ, застосовується обмеження, яке $(\alpha * \beta^2 * \gamma^2) \approx 2$. Масштабування не змінює операцій з шарами, тому краще спочатку мати хорошу базову мережу, а потім масштабувати її за різними розмірностями, використовуючи запропоноване складне масштабування. Автори отримали свою базову мережу, виконавши Neural Architecture Search (NAS), який оптимізує як точність, так і FLOPS. Архітектура схожа на M-NASNet, оскільки її було знайдено за допомогою подібного простору пошуку. Структуру блоків мережі показано на рисунку 2.16 нижче:

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	28×28	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Рисунок 2.16 – Структура EficcientNet-B0

Тепер маючи базову мережу, ми можемо шукати оптимальні значення для наших параметрів масштабування. У нас є всього чотири параметри для пошуку: α, β, γ і ϕ Для того, щоб зменшити простір пошуку та зробити операцію пошуку менш затратною використаємо наступний підхід:

1) Фіксуємо $\phi = 1$, припускаючи, що доступно вдвічі більше ресурсів, та виконуємо невеликий жадібний пошук для α, β та γ. Для базової мережі B0 виявилося, що оптимальними є значення α = 1,2, β = 1,1 та γ = 1,15 такі, що $\alpha * \beta^2 * \gamma^2 \approx 2$
2) Фіксуємо отримані α, β і γ як константи і експериментуємо з різними значеннями ϕ. Різні значення ϕ утворюють сім'ю мереж EfficientNets B1-B7.

Головною перевагою використання EfficientNet є те, що при відносно невеликій кількості параметрів, яка потребує меншої кількості обчислень, вона показує таку ж точність як і більш складні моделі.

2.3 Метрики якості прогнозу

Для подальшого порівняння представлених методів вирішення поставленої задачі, необхідно визначити метрики за якими будуть оцінюватись навчені моделі. Оскільки перед нами задача класифікації де кожному вхідному

зразку присвоєно сукупність міток, то метрики за якими оцінюють такі моделі дещо відрізняються від метрик бінарної або мультикласової класифікації. Виділяють дві категорії метрик: метрики на основі зразків та метрики на основі міток.

Example-based метрики призначені для обчислення середньої різниці між справжніми мітками та передбачуваними мітками для кожної точки навчальних даних, усередненої за всіма навчальними прикладами в наборі даних.

1) Exact Match Ratio(EMR)

$$EMR = \frac{1}{n}\sum_{i=1}^{n}\left[I(y^{(i)} == \hat{y}^{(i)}\right]$$

де *n* - кількість зразків в навчальному наборі,

$y^{(i)}$ - істинні мітки для *i*-го зразка,

$\hat{y}^{(i)}$ - прогнозні мітки для *i*-го зразка.

Метрика оцінки точного співвідношення розширює концепцію точності від задачі класифікації з однією міткою до проблеми класифікації з багатьма мітками. Одним з недоліків використання *EMR* є те, що вона не враховує частково правильні мітки.

2) Hamming Loss

$$Hamming\ Loss = \frac{1}{nL}\sum_{i=1}^{n}\sum_{j=1}^{L}\left[I(y_j^{(i)} \neq \hat{y}_j^{(i)})\right]$$

де *n* - кількість зразків в навчальному наборі,

L – кількість можливих міток $y_j^{(i)}$ - істинні мітки для *j*-го класу *i*-го зразка,

$\hat{y}_j^{(i)}$ - прогнозні мітки для *j*-го класу *i*-го зразка.

Hamming Loss обчислює частку неправильно передбачених міток до загальної кількості міток. Для multi-label класифікації ми обчислюємо кількість False Positive і False Negative міток для одного зразка, а потім усереднюємо його за загальною кількістю записів у навчальному наборі.

3) Example-Based Accuracy

$$Accuracy(A) = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|y^{(i)} \wedge \hat{y}^{(i)}\right|}{\left|y^{(i)} \vee \hat{y}^{(i)}\right|}$$

де n - кількість зразків в навчальному наборі,

$y^{(i)}$ - істинні мітки для i-го зразка,

$\hat{y}^{(i)}$ - прогнозні мітки для i-го зразка.

Accuracy для навчального екземпляру визначається як частка передбачуваних правильних міток до загальної кількості міток для цього навчального екземпляру. Загальна точність - це середнє значення точності між навчальними прикладами.

4) Example-Based Precision

$$Precision(P) = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|y^{(i)} \wedge \hat{y}^{(i)}\right|}{\left|\hat{y}^{(i)}\right|}$$

де n - кількість зразків в навчальному наборі,

$y^{(i)}$ - істинні мітки для i-го зразка,

$\hat{y}^{(i)}$ - прогнозні мітки для i-го зразка.

Точність на основі прикладів визначається як частка передбачуваних правильних міток до загальної кількості передбачуваних міток, усереднена по всім екземплярам.

На відміну від example-based metrics, метрики на основі міток оцінюють кожну мітку окремо, а потім усереднюють результат по всіх мітках. Це означає, що будь-яка метрика використовується для бінарної класифікації, може бути використана як метрика на основі міток. Найчастіше вимірюють значення accuracy, presicion, recall та F1-score. Ці показники можна обчислити за окремими мітками, а потім усереднити по всіх класах. Це називається Macro Averaging. Крім того, ми можемо обчислити ці показники глобально для всіх екземплярів та усіх міток класів. Це називається Micro Averaging.

1) Macro Averaged metrics

$$A^j_{Macro} = \frac{\sum_{i=1}^{n}[y_j^{(i)} \wedge \hat{y}_j^{(i)}]}{\sum_{i=1}^{n}[y_j^{(i)} \vee \hat{y}_j^{(i)}]}, \qquad Accuracy_{Macro} = \frac{1}{k}\sum_{j=1}^{k} A^j_{Macro}$$

$$P^j_{Macro} = \frac{\sum_{i=1}^{n}[y_j^{(i)} \wedge \hat{y}_j^{(i)}]}{\sum_{i=1}^{n}[\hat{y}_j^{(i)}]}, \qquad Precision_{Macro} = \frac{1}{k}\sum_{j=1}^{k} P^j_{Macro}$$

$$R^j_{Macro} = \frac{\sum_{i=1}^{n}[y_j^{(i)} \wedge \hat{y}_j^{(i)}]}{\sum_{i=1}^{n}[y_j^{(i)}]}, \qquad Recall_{Macro} = \frac{1}{k}\sum_{j=1}^{k} R^j_{Macro}$$

$$F1_{Macro} = \frac{2 * precision_{Macro} * recall_{Macro}}{(precision_{Macro} + recall_{Macro})}$$

2) Micro Averaged metrics

$$Accuracy_{Micro} = \frac{\sum_{j=1}^{k}\sum_{i=1}^{n}[y_j^{(i)} \wedge \hat{y}_j^{(i)}]}{\sum_{j=1}^{k}\sum_{i=1}^{n}[y_j^{(i)} \vee \hat{y}_j^{(i)}]}$$

$$Precision_{Micro} = \frac{\sum_{j=1}^{k}\sum_{i=1}^{n}[y_j^{(i)} \wedge \hat{y}_j^{(i)}]}{\sum_{j=1}^{k}\sum_{i=1}^{n}\hat{y}_j^{(i)}}$$

$$Recall_{Micro} = \frac{\sum_{j=1}^{k}\sum_{i=1}^{n}[y_j^{(i)} \wedge \hat{y}_j^{(i)}]}{\sum_{j=1}^{k}\sum_{i=1}^{n}y_j^{(i)}}$$

$$F1_{Micro} = \frac{2 * precision_{Micro} * recall_{Micro}}{(precision_{Micro} + recall_{Micro})}$$

де n - кількість зразків в навчальному наборі,

k – кількість можливих міток $y_j^{(i)}$ - істинні мітки для j-го класу i-го зразка,

$\hat{y}_j^{(i)}$ - прогнозні мітки для j-го класу i-го зразка.

2.4 Порівняння існуючих методів

Для порівняння роботи класичних класифікаторів використаємо набір даних FLOWER17, наданий Оксфордським університетом. Цей датасет складається з зображень 17 різних класів квітів. За допомогою дескрипторів бібліотеки OpenCV дістанемо ознаки для кожного екземпляра. Далі набір поділений на навчальний та тестовий у відношенні 90/10. Реалізовуючи низку методів класифікації, які були описані раніше, отримано наступні результати для кросс-валідації з 5 вікнами(рис. 2.17, зліва шкала точності прогнозу):

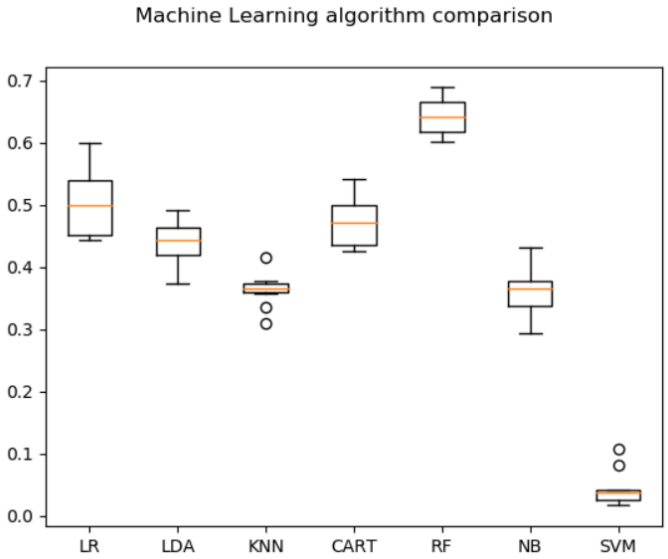


Рисунок 2.17 – Результати роботи класифікаторів

Найкраще показав себе метод Random Forest, але треба зауважити що набір даних мав всього 1360 записів. Для порівняння роботи згорткових мереж використовують набагато більші датасети з величезним різноманіттям класів. Серед них MNIST, CIFAR, ImageNet. Багато з них беруть участь у змаганнях ILSVRC(ImageNet Large Scale Visual Recognition Challenge). ResNet показує кращі результати ніж VGG, та використовує менше ресурсів, у зв'язку з меншою кількістю параметрів(рис. 2.18).

method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

Рисунок 2.18 – Процент похибки моделей. Top-5 похибка для тестового набору ImageNet

Натомість моделі DenseNet та EfficientNet показують більшу швидкодію меншу похибку ніж ResNet(рис. 2.19, 2.20).

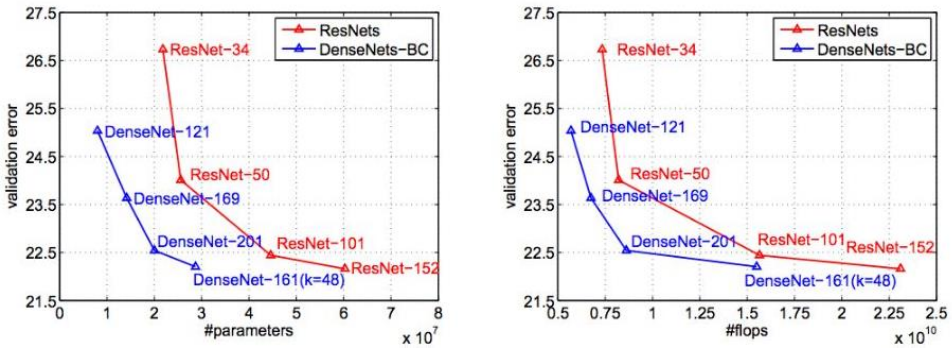


Рисунок 2.19 – Результати роботи DenseNet та ResNet з набором ImageNet

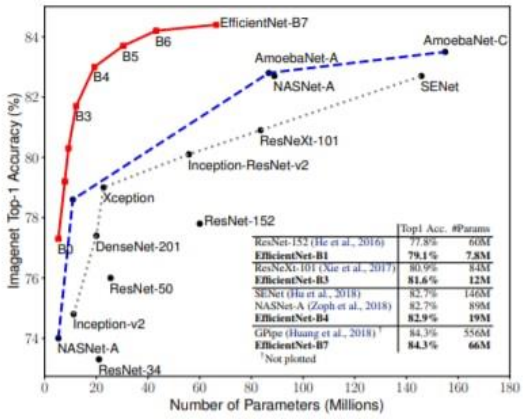


Рисунок 2.20 – Результати роботи EfficientNet.

Загалом можна зробити висновок, що згорткові нейронні мережі працюють швидше на великих наборах даних, показують вищу точність класифікації, порівняно з базовими моделями. Саме тому для вирішення поставленої задачі розпізнавання патологій, будемо використовувати CNN.

2.5 Висновки за розділом 2

В цьому розділі було проаналізовано та описано математичні основи методів розпізнавання для задач комп’ютерного зору. Розглянуто структуру основних алгоритмів класифікації зображень. Умовно можна поділити підходи на: використання класичних моделей машинного навчання та використання нейронних мереж. Серед алгоритмів, які є канонічними та застосовуються для подібного роду задач можна виділити: логістичну регресію, випадковий ліс,

класифікатор з використанням опорних векторів. Натомість, серед багатьох моделей глибинних нейронних мереж, саме згорткові нейронні мережі найчастіше застосовуються для задач класифікації зображень. В розділі детально описано архітектуру та принцип роботи CNN. Також зроблено огляд структури та головних ідей таких готових моделей: VGG, ResNet, DenseNet та EfficientNet. Визначено основні метрики оцінки якості роботи multi-label класифікатора. У ході порівняння розглянутих підходів, було показано ефективність застосування саме згорткових мереж, замість класичних методів. На сьогоднішній день, CNN показують блискучі результати для великих та складних наборів даних з безліччю різноманітних класів.

РОЗДІЛ 3
АРХІТЕКТУРА ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ ПОБУДОВАНОГО ПРОГРАМНОГО ПРОДУКТУ

3.1 Вибір мови програмування та фреймворків

Для реалізації системи розпізнавання патологій рослин будемо використовувати згорткові нейронні мережі. В якості мови програмування для реалізації поставленої задачі було обрано Python, версія 3.8. Основними перевагами використання саме цієї мови є швидкість написання готового продукту, простота синтаксису та підтримка великої кількості бібліотек, необхідних для роботи. Окрім цього, Python часто використовують в задачах машинного навчання та комп'ютерного зору. Для успішного створення програмного продукту нам будуть потрібні засоби для обробки даних, навчання моделі, оцінки якості побудованих рішень. Для цього ми використаємо наступні бібліотеки та фреймворки готових рішень, доступні для обраної мови програмування:

1) OpenCV - це кросплатформна бібліотека, яка в основному призначена для задач комп'ютерного зору в режимі реального часу. Основна увага приділяється обробці зображень, захопленню та аналізу відео, включаючи такі функції, як виявлення обличчя та виявлення об'єктів.
2) Tensorflow - це фреймворк з відкритим кодом для машинного навчання. Представляє собою комплексну та гнучку екосистему інструментів, бібліотек та інших ресурсів, які забезпечують робочі процеси за допомогою високоякісних API. Структура пропонує різні рівні концепцій, щоб ви могли вибрати той, який вам потрібен для побудови та впровадження моделей машинного навчання. Зокрема, Keras - це бібліотека нейронних мереж високого рівня, яка працює на вершині TensorFlow, CNTK та Theano. Використання Keras у процесі deep learning

дозволяє легко і швидко створювати прототипи, а також безперебійно працювати на центральному та графічному процесорі.

3) Scikit-learn (Sklearn) - це бібліотека на Python, яка підтримує безліч supervised та unsupervised алгоритмів навчання. Містить багато ефективних інструментів для машинного навчання та статистичного моделювання, включаючи класифікацію, регресію, кластеризацію. Окрім цього дозволяє виконувати preprocessing, оцінку якості прогнозування, пропонує різні методи вибору моделі та найкращих параметрів навчання.

Для подальшої розробки, окрім описаних засобів, використано наступні бібліотеки: Albumentations - для аугментації зображень, NumPy - для роботи з масивами, SciPy - для обчислень, Pandas – для роботи з базами даних та таблицями, Matplotlib – для візуалізації результатів дослідження.

3.2 Аналіз та попередня обробка даних для дослідження

В першому розділі було встановлено, що вхідними даними для дослідження є високоякісні зображення листків яблучних дерев. Оскільки обробка зображень розмірністю 2672×4000 займає багато часу, та часто потребує надлишкових ресурсів, при цьому не сильно покращуючи роботу моделі, розмір всіх вхідних зображень змінено до 600×600. Також в ході роботи з початковими даними, було виявлено, що в наборі присутні 56 пар зображень-дублікатів. На однакових зображення були відмічені різні мітки захворювань або зображення були просто продубльовані під різними іменами(рис. 3.1). В подальшому цей факт може погіршити роботу класифікатора, тому виконаємо наступну операцію: залишаємо одне зображення з пари, якщо вони помічені однаково, видаляємо пару дублікатів, якщо вони помічені по різному. Таким чином було видалено 77 зображень.

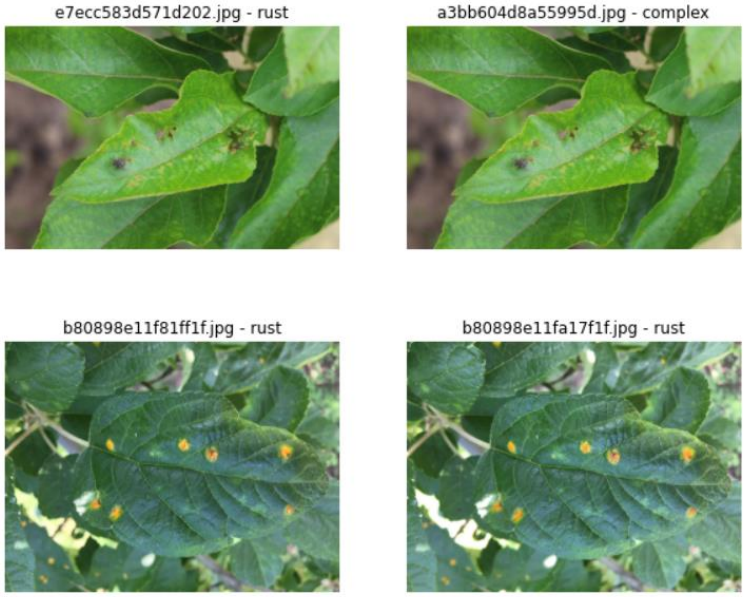


Рисунок 3.1 – Приклад зображень-дублікатів

Попередній аналіз показав що розподіл міток різних захворювань в наборі є дуже нерівномірним. Це пов’язано з тим, що в початковому наборі наявність двох чи більше хвороб на одному листку відмічена окремим класом, наприклад «scab frog_eye_leaf_spot complex». Для вирішення проблеми нерівномірності розподілу міток та для приведення датасету до класичної задачі multi-label класифікації було виконано алгоритм One-Hot Encoding. Принцип його роботи полягає в створенні окремих атрибутів(колонок) відповідно для кожного класу захворювань, та кодування зображень 0 або 1, у разі наявності відповідної патології. В нашому випадку окремих категорій усього 6: rust, scab, complex, frog_eye_leaf_spot, powdery_mildew, healthy. Після проведення цієї процедури отримуємо наступний результат(рис. 3.2, 3.3).

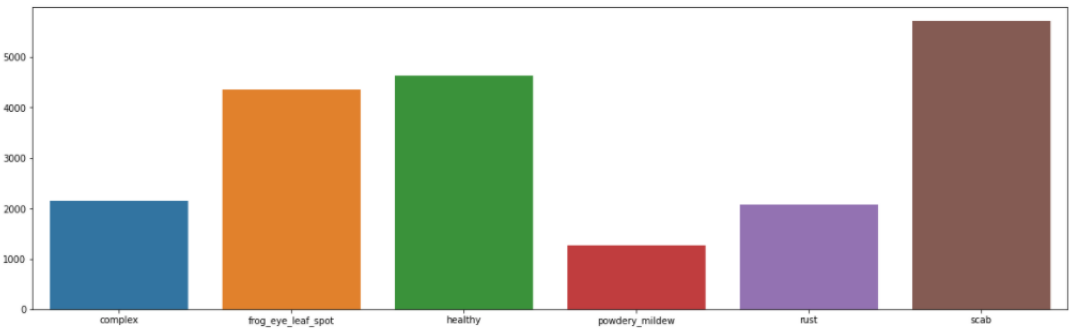


Рисунок 3.2 – Кількість представників окремих класів після кодування

	complex	frog_eye_leaf_spot	powdery_mildew	rust	scab	healthy
image						
800113bb65efe69e.jpg	0	0	0	0	0	1
8002cb321f8bfcdf.jpg	1	1	0	0	1	0
80070f7fb5e2ccaa.jpg	0	0	0	0	1	0
80077517781fb94f.jpg	0	0	0	0	1	0
800cbf0ff87721f8.jpg	1	0	0	0	0	0

Рисунок 3.3 – Таблиця з мітками після кодування

Для того щоб система розпізнавання гарно працювала в реальних умовах, навчальний набір має бути більш різноманітним. Він має включати в себе такі варіації одного і того ж зображення: зміна точки зору, освітленості, масштабу, деформацію, тощо. Це дозволяє натренувати модель більш стійку до втрати якості зображень та пристосовану до роботи в реальному часі. Для цього необхідно використовувати аугментації або трансформації зображень. Серед основних прийомів можна виділити такі: зміна кольору, яскравості/контрастності, геометричні перетворення, додавання різного роду шумів.

Щоб використовувати даний підхід, в цій роботі було підключено бібліотеку Albumentations. Вона містить широкий спектр функцій-трансформацій та пристосована до роботи з фреймворками глибокого навчання. Наведемо приклад роботи для одного зображення, тут чорні квадрати на зображенні – це гаусівський шум який додається до значень пікселів і при виході за мінімальні та максимальні значення, установлює граничні значення: 255(білий колір) та 0(чорний колір). В нашому випадку будемо застосовувати випадкове перетворення із набору методів бібліотеки, для кожного з яких визначена ймовірність настання. Приклад роботи деяких з них показаний на рисунку 3.4.

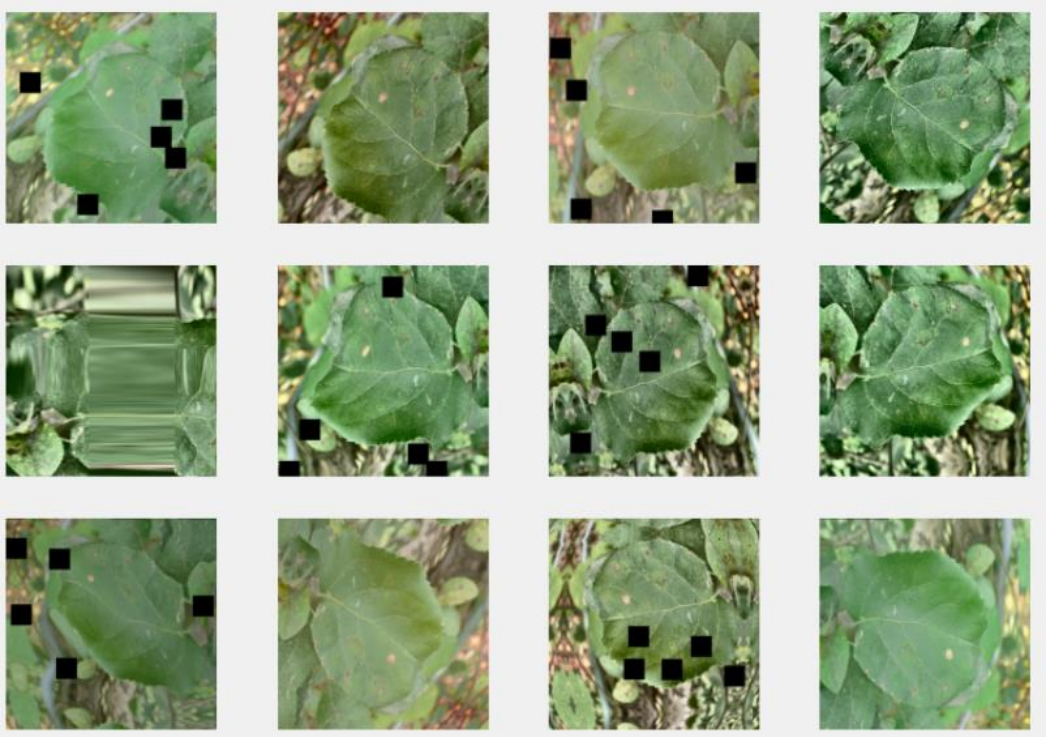


Рисунок 3.4 – Приклад аугментацій одного зображення

Для ефективної роботи з даними, в тому числі і на TPU(Tensor Processing Unit), TensorFlow надає формат TFRecord. Це простий формат для зберігання послідовності двійкових записів. Він дозволяє зберігати дані про зображення та мітки до нього в компактному форматі. Використання TFRecord значно пришвидшує обробку та зчитування інформації, що позитивно впливає на час навчання моделі. Саме тому в розробленій системі, на вхід моделі будуть подаватись файли в форматі TFRecord.

3.3 Архітектура системи

Програмний продукт являє собою натреновану модель з визначеною структурою, яка може бути інтегрована в готові рішення та продукти для зйомки нових зображень в реальному часі. Наприклад, це можуть бути автоматизовані дрони або камери, які переміщаються по саду та знімають стан листя дерев. В цей час система в автоматичному режимі розпізнає наявність захворювання, далі

ці дані можуть передаватись до центру управління або зберігатись безпосередньо на пристрої. Подібне рішення дозволяє автоматизувати процес діагностики насаджень та підвищити ефективність виробництва.

Як вже було сказано раніше, модель представлена у вигляді згорткової нейронної мережі. Для пришвидшення процесу навчання доцільно було обрати відому, натреновану мережу. Серед можливих варіантів готових рішень було розглянуто використання моделей EfficientNetB4 та DenseNet169. EfficientNetB4 порівняно з іншими, у зв'язку з меншою кількістю параметрів, вона потребує менше часу на навчання та має досить високу точність результатів. У якості початкової матриці вагів було обрано noisy-student weights. Noisy Student Training – це semi-supervised навчання, яке добре працює, навіть коли маркованих даних багато. Цей підхід досягає 88,4% точності top-1 на ImageNet, що на 2,0% краще, ніж ультрасучасна модель, яка вимагає 3,5 млрд. слабо розмічених зображень Instagram. На наборах для перевірки стійкості Noisy Student Training покращує точність ImageNet-A top-1 з 61,0% до 83,7%, зменшує середню похибку корупції ImageNet-C з 45,7 до 28,3 та зменшує середню частоту перевертання ImageNet-P з 27,8 до 12,2. Така модель розширює ідею самонавчання та дистиляції з використанням рівних чи більших моделей noisy-student, що додається до студента під час навчання. На ImageNet спочатку тренуємо модель EfficientNet на маркованих зображеннях і використовуємо її як викладача для створення псевдоміток для 300 млн. немаркованих зображень. Потім навчаємо більшу EfficientNet як студентську модель на поєднанні розмічених та псевдомічених зображень. Процес повторюється, використовуючи студента як вчителя. Під час навчання студента додаються такі шуми, як відсівання, стохастична глибина та збільшення кількості даних через Rand-Augment, щоб студент узагальнював краще за вчителя. DenseNet169, в свою чергу, була обрана, оскільки, незважаючи на глибину в 169 шарів, має небагато параметрів в порівнянні з іншими моделями, її архітектура добре справляється з проблемою затухаючого градієнта.

Pre-trained EfficientNetB4 була дещо змінена для поставленої задачі. Останній повнозв'язний шар розмірністю 1000×1 був замінений на шар 5×1 для передбачення п'яти класів хвороб, у випадку якщо зображення не належить до жодної з категорій, воно маркується міткою healthy. У якості функції-активації повнозв'язного шару обрано сигмоїд, тому що він підтримує задачі multi-label, в той час як softmax орієнтована на визначення одного лиш класу. Отже, фінальна структура використаної мережі EfficientNetB4 має вигляд(рис. 3.5):

```
Model: "EfficientNetB4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
efficientnet-b4 (Functional) (None, 1792)              17673816
_________________________________________________________________
dense_top (Dense)            (None, 5)                 8965
_________________________________________________________________
activation (Activation)      (None, 5)                 0
=================================================================
Total params: 17,682,781
Trainable params: 17,557,581
Non-trainable params: 125,200
_________________________________________________________________
```

Рисунок 3.5 – Архітектура готової мережі EfficientNetB4

Pre-trained DenseNet169 також була доповнена для вирішення поставленої задачі. Замість останнього повнозв'язного шару було додано шари GlobalAveragePooling2D та Dropout. Окрім цього, додано три повнозв'язні шари: 128×1, 64×1, 5×1. У якості функції-активації повнозв'язного шару також обрано сигмоїд. Отже, останні шари налаштованої мережі DenseNet169 мають вигляд(рис. 3.6):

```
conv5_block32_concat (Concatena (None, 7, 7, 1664)   0           conv5_block31_concat[0][0]
                                                                  conv5_block32_2_conv[0][0]
_____________________________________________________________________________________________
bn (BatchNormalization)         (None, 7, 7, 1664)   6656        conv5_block32_concat[0][0]
_____________________________________________________________________________________________
relu (Activation)               (None, 7, 7, 1664)   0           bn[0][0]
_____________________________________________________________________________________________
tf.math.reduce_mean_12 (TFOpLam (None, 1664)         0           relu[0][0]
_____________________________________________________________________________________________
dense_36 (Dense)                (None, 128)          213120      tf.math.reduce_mean_12[0][0]
_____________________________________________________________________________________________
dropout_12 (Dropout)            (None, 128)          0           dense_36[0][0]
_____________________________________________________________________________________________
dense_37 (Dense)                (None, 64)           8256        dropout_12[0][0]
_____________________________________________________________________________________________
dense_38 (Dense)                (None, 5)            325         dense_37[0][0]
=============================================================================================
Total params: 12,864,581
Trainable params: 12,706,181
Non-trainable params: 158,400
```

Рисунок 3.6 – Останні шари готової мережі DenseNet169

Для навчання обрано стратегію ансамблю з 5 незалежних крос-валідаційних вікон(folds). Для цього, на початку розділимо весь набір зображень на 5 частин, що не перетинаються, зберігаючи розподіл кожного з класів в кожному наборі. Цей підхід називають стратифікацією(рис.3.7).

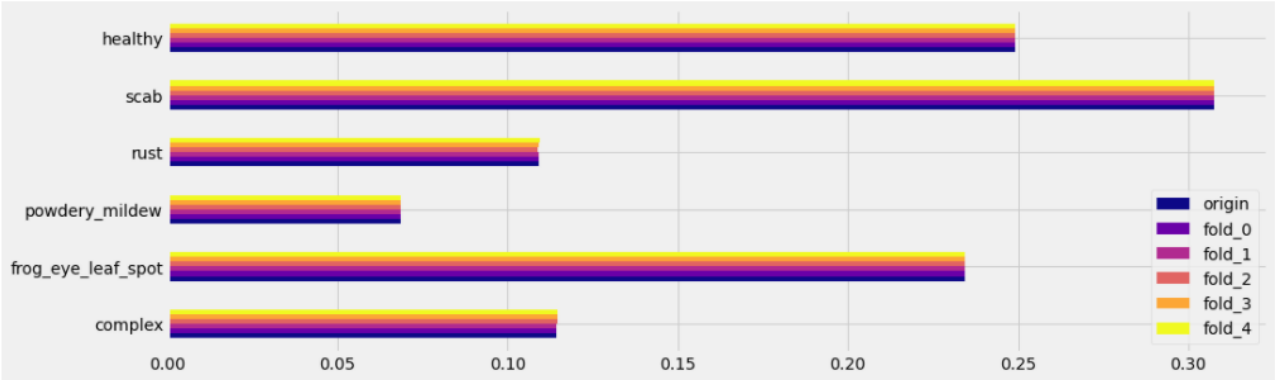


Рисунок 3.7 – Стратифікація на 5 частин

Далі, до кожного з наборів застосуємо аугментації. Виконаємо цю процедуру 4 рази. Таким чином кожного разу будуть створюватись різні випадкові трансформації зображень. В результаті отримуємо 4 датасети різних перетворень 5 наборів. Також використаємо вхідний, не трансформований набір, який таким самим способом розділений на 5 частин, для валідації результатів навчання. Всі дані для зручності обробки перекодовано в формат .tfrec по 16 екземплярів. Тобто кожне з п’яти розбиттів вхідного набору даних містить файли .tfrec, кожний з яких містить інформацію про 16 окремих зображень.

Процес навчання формується таким чином: здійснюється крос-валідація по кожному з розбиттів. Наприклад, у якості навчального набору береться 1, 2, 3, 4 папка з 4 аугментованих датасетів, а у якості перевірочної вибірки береться п’ята папка з вихідного набору(рис. 3.8). Необроблений початковий набір завжди використовується для валідації, тому модель не навчається на вхідних даних, лише на трансформованих. Таким чином проводиться 5 етапів навчання.

```
0
[0 2 3 4] [1]
1
[0 1 2 3] [4]
2
[0 1 3 4] [2]
3
[1 2 3 4] [0]
4
[0 1 2 4] [3]
```

Рисунок 3.8 – Розбиття при крос-валідації(справа - перевірочна папка)

У якості метрики оцінки якості моделі обрано Macro F1-score, але також обчислюється BinaryAccuracy, що показує частоту співпадіння прогнозів з двійковими мітками(0 або 1) класів. Оптимізація проводиться алгоритмом Adam, початкова швидкість навчання 0.001, кількість епох – 100. У якості callbacks для навчання використовуються два підходи: EarlyStopping та ReduceLROnPlateau. EarlyStopping застосовується у випадку, коли протягом 5 епох, показник якості, в нашому випадку F1-score не перевищує максимально зафіксований результат. При цьому запам’ятовуються значення матриці вагів, за яких був отриманий найкращий результат. Завдяки цьому критерію, кількість необхідних для навчання епох для одного етапу крос-валідації буде значно меншою за 100. ReduceLROnPlateau спрацьовує коли модель перестає вчитись, та починає показувати гірші результати. Для виходу із стану стагнації знижують швидкість навчання. В нашому випадку, якщо протягом 2 епох, модель не покращує якість, швидкість навчання зменшується в 10 разів. Таким чином проводиться навчання окремих моделей по кожному з 5 проміжків крос-валідації. Отримані значення матриці вагів для кожної з 5 мереж зберігаються в окремі файли. Так F1-score для всіх отриманих моделей має наступні графіки(рис.3.9, 3.10):

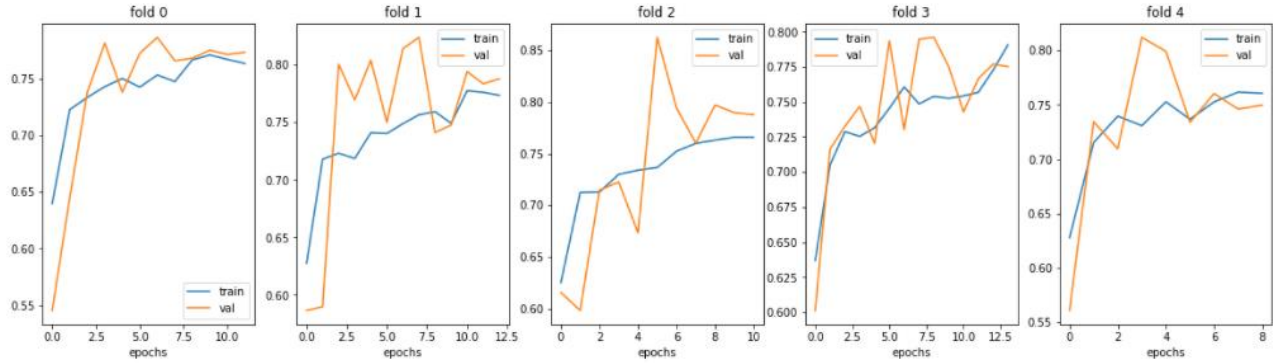


Рисунок 3.9 – Графіки зміни значень F1-score протягом навчання (EfficientNetB4)

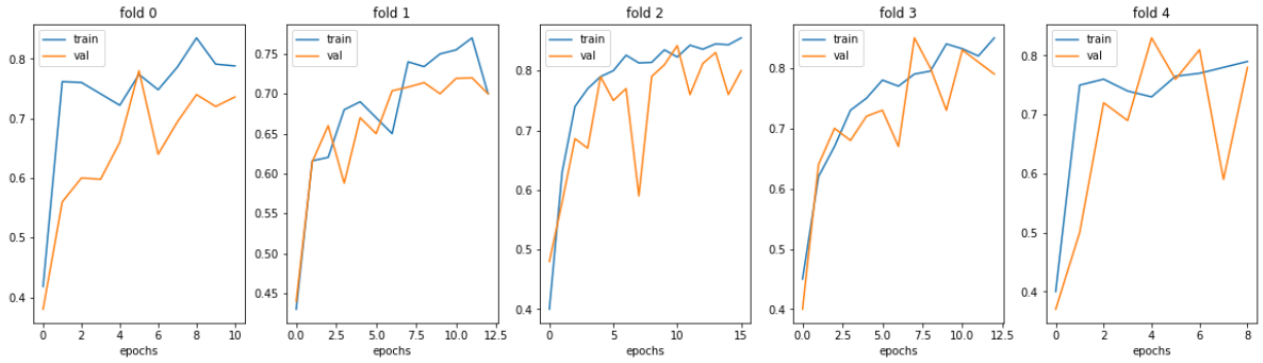


Рисунок 3.10 – Графіки зміни значень F1-score протягом навчання (DenseNet169)

Максимальні значення оцінок F1-score для валідаційної вибірки наведені у таблиці 3.1

Таблиця 3.1 – Результати навчання для валідаційної вибірки

Модель	fold 0	fold 1	fold 2	fold 3	fold 4	Average
EfficientNetB4	0.7866	0.8239	0.8627	0.7960	0.8121	0.8163
DenseNet169	0.7806	0.7294	0.8421	0.8542	0.8317	0.8076

У середньому значення F1-score для валідаційної вибірки у випадку використання EfficientNetB4 дорівнює 0,816. Отже, аналізуючи результати, можна сказати, що модель EfficientNetB4 показує вищу точність. Саме тому для подальшої роботи було вирішено використовувати саме її.

Окрім обчислення значень метрики, в ході навчання також записуються прогнозні значення ймовірностей для кожної з моделей, відповідно до оптимальних значень параметрів. Таким чином в результаті отримуємо набір, де для кожного вихідного зображення обчислена ймовірність попадання до того чи іншого класу патологій. Дана інформація необхідна для встановлення оптимальних порогових значень для входження до кожної з категорій. Отримані значення порівнюють з початковою таблицею train.csv після перекодування(окрім колонки healthy), намагаючись максимізувати значення F1-score для кожного з класів шляхом перебору. Таким чином отримуємо наступні значення для порогів для 5 класів хвороб: complex = 0.37, frog_eye_leaf_spot = 0.41, powdery_mildew = 0.17, rust = 0.36, scab = 0.57(рис. 3.11).

```
complex: 0.7165 >>> 0.7244 (0.37)
frog_eye_leaf_spot: 0.8852 >>> 0.8858 (0.41)
powdery_mildew: 0.9478 >>> 0.9561 (0.17)
rust: 0.9317 >>> 0.9334 (0.36)
scab: 0.9109 >>> 0.9120 (0.57)

mean score: 0.8784 >>> 0.8823
```

Рисунок 3.11 – Налаштування порогових значень

Тобто якщо для нового зображення, прогнозовані ймовірності кожного з класів не перетинають поріг, екземпляр отримує мітку healthy.

3.4 Практичні результати

Оскільки вхідні дані знаходяться у відкритому доступі в рамках наукового змагання на ресурсі Kaggle, то є можливість перевірити якість моделі на нерозміченому тестовому наборі та скористатись автоматичною системою підрахунку результату. Важливо відзначити, що в якості метрики для перевірки також використовується середнє значення F1-score, це ще раз показує

доцільність використання саме цього показника для оцінки роботи. Тестовий набір містить наступні 3 зображення(рис. 3.12):

Рисунок 3.12 – Тестовий набір

Для роботи з тестовими даними, для початку змінимо їх розмір до 600×600 та перекодуємо до формату .tfrec. Далі завантажуються значення вагів для кожної з 5 моделей, робиться передбачення і береться середнє значення отриманих ймовірностей. Таким чином для тестового набору зроблено прогноз(рис. 3.13):

image	complex	frog_eye_leaf_spot	powdery_mildew	rust	scab
85f8cb619c66b863.jpg	0.490224	0.724904	0.000142	0.001134	0.699495
ad8770db05586b59.jpg	0.182869	0.915205	0.000081	0.001823	0.212230
c7b03e718489f3ca.jpg	0.316361	0.049370	0.000496	0.020903	0.588541

Рисунок 3.13 – Прогноз ймовірностей для тестового набору

Використовуючи обчислені раніше порогові значення входження до окремих класів, тестовим зображенням можна присвоїти наступні мітки(рис. 3.14):

	image	labels
0	85f8cb619c66b863.jpg	complex frog_eye_leaf_spot scab
1	ad8770db05586b59.jpg	frog_eye_leaf_spot
2	c7b03e718489f3ca.jpg	scab

Рисунок 3.14 – Передбачені мітки тестових зображень

Отримані результати завантажуємо в систему перевірки та отримуємо результат роботи побудованої моделі розпізнавання для нових даних. В нашому випадку значення оцінки дорівнює 0,813. Це досить непоганий результат для подібних задач, але отриману систему звісно можна покращити. Спробувати розширити датасет більшою кількістю аугментацій, збалансувати класовий розподіл. Також можна спробувати інші методи оптимізації та налаштування швидкості навчання.

3.5 Висновки до розділу 3

В ході розробки було проведено три основних етапи: передобробка даних, навчання моделі, постпроцесинг результатів та перевірка на тестових зображеннях. На першому етапі розробки системи розпізнавання патологій рослин вхідний набір зображень був приведений до зручного для подання на вхід мережі вигляду. Для збільшення швидкості навчання було зменшено розмір ображень, видалено дублікати. Також проведено перекодування міток за принципом One-Hot. Для розширення наявного датасету сукупністю аугментацій наявних зображень були застосовані засоби бібліотеки Albumentations. Безпосередньо для навчання моделі з використанням TensorFlow вхідні дані перекодовано в формат TFRecord.

Перед навчанням моделі було вирішено використовувати принцип незваженого ансамблю з 5 частин. Вхідний датасет та 4 набори з трансформаціями були розділені на 5 крос-валідаційних частин. Це дозволило навчатись поетапно та в цілому пришвидшити цей процес. У якості базової мережі було обрано EfficientNetB4 та DenseNet169, оскільки вони мають меншу кількість параметрів для навчання, ніж інші моделі та показують більшу швидкодію. Повнозв’язний шар побудованої нейронної мережі здійснює класифікацію за п’ятьма наявними категоріями хвороб. Моделі навчаються на наборі з трансформованими зображеннями та проходить валідацію на

початковому датасеті. Метрикою якості було обрано середнє значення F1-score для кожного з класів. Моделі показують гарні результати як на навчальному наборі, так і на валідаційному, це означає що мережа уникає перенавчання. Це досягається шляхом корегування швидкості навчання. В ході навчання було визначено, що кращі результати показує безпосередньо мережа з базовою моделлю EfficientNetB4, тому для подальшого дослідження вирішено використовувати дане рішення. В результаті отримано 5 моделей та значення прогнозу для всього вихідного датасету.

На етапі постпроцесингу, отримані дані використовуються для встановлення оптимальних порогів відсікання. Це відбувається шляхом перебору з метою максимізувати значення метрики оцінки. Прогнозування міток на нових даних побудоване наступним чином: розмір тестових даних змінюється до 600x600, далі вони приводяться до формату .tfrec і подаються на вхід п’ятьом різним моделям. Отримані значення ймовірностей усереднюють і, враховуючи встановлені пороги відсікання, формуються мітки класів. У випадку якщо не спрогнозовано жодну з 5 категорій хвороб, екземпляр відноситься до класу healthy. Також був проведений експеримент з використанням тестових даних в рамках змагань на ресурсі Kaggle, розроблена система показала гарні результати, передбачення для нових зображень отримало оцінку F1-score рівну 0,813.

РОЗДІЛ 4
ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

4.1 Постановка задачі проектування

У даному розділі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи розпізнавання патологій рослин. Фактичний аналіз представляє собою аналіз функцій програмного продукту, призначених для обробки, аналізу та прогнозування даних про захворювання рослин, тому що кожен компонент має задовольняти вимоги рішень для реалізації та проектування системи в цілому.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп'ютерах, а також на серверах з доступом до мережі Інтернет;
- швидкість та простота встановлення на апаратну частину системи;
- зручність та зрозумілість для користувача;
- швидкість обробки зображень та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який вирішує задачу розпізнавання патологій рослин, виходячи з зображень листя. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування;

F_2 – вибір фреймворку машинного навчання;

F_3 – вибір середовища розробки.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) Python

б) C++

Функція F_2:

а) Tensorflow;

б) PyTorch

Функція F_3:

а) Jupyter Notebook;

б) Visual Studio.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

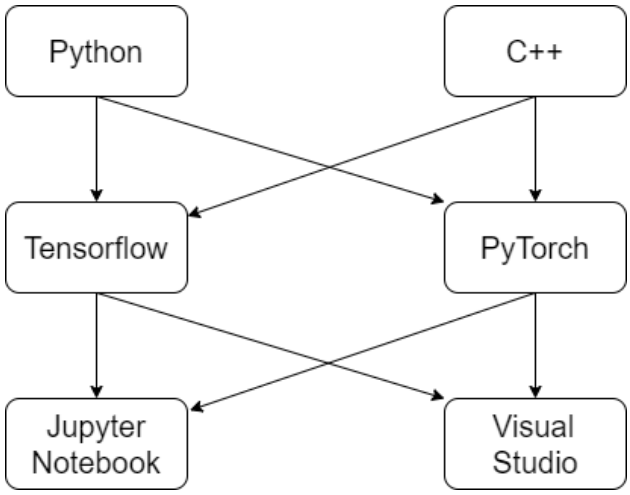


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 4.1).

Таблиця 4.1 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*А*	Наявність високорівневих бібліотек, гнучкий інструментарій	Менша швидкість виконання програми
	Б	Швидкість виконання коду	Складність розробки та підтримки
F_2	*А*	Пристосований до систем реального часу, швидкий для інтеграції	Не підтримується багатьма мовами
	Б	Підтримує обидві платформи, швидкість розробки, низькорівневий доступ	Додатковий час на інсталяцію та вивчення
F_3	*А*	Підтримується багатьма мовами програмування, легко запускається на будь-якому сервері	Відсутня можливість роботи без інтернету
	Б	Багато інструментів, безпечна	Підтримує одночасно лише одну мову програмування

Робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1:

Перевагу віддаємо швидкості розробки, простоті використання та наявності стандартних бібліотек для обчислення. Для спрощення роботи по написанню коду варіант Б має бути відкинутий.

Функція F_2:

Обидва варіанти можна використовувати в розробці.

Функція F_3:

Віддаємо перевагу варіанту А в разі вибору мови програмування Python.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

F_1а – F_2а – F_3а

F_1а – F_2б – F_3а

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів ПП

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня. Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- *X1* – час розробки;

- *X2* – швидкодія мови програмування;
- *X3* – об’єм пам’яті для збереження даних;
- *X4* – необхідний ресурс оперативної пам’яті;
- *X5* – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у таблиці 4.2.

Таблиця 4.2 - Основні параметри ПП

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Час розробки	X1	людина*год	120	90	60
Швидкодія мови програмування	X2	оп/мс	10000	14000	19000
Об’єм пам’яті для збереження даних	X3	гб	20	12	4
Необхідний ресурс оперативної пам’яті	X4	мб	2000	1500	1100
Потенційний об’єм програмного коду	X5	кількість рядків коду	4000	2500	1000

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

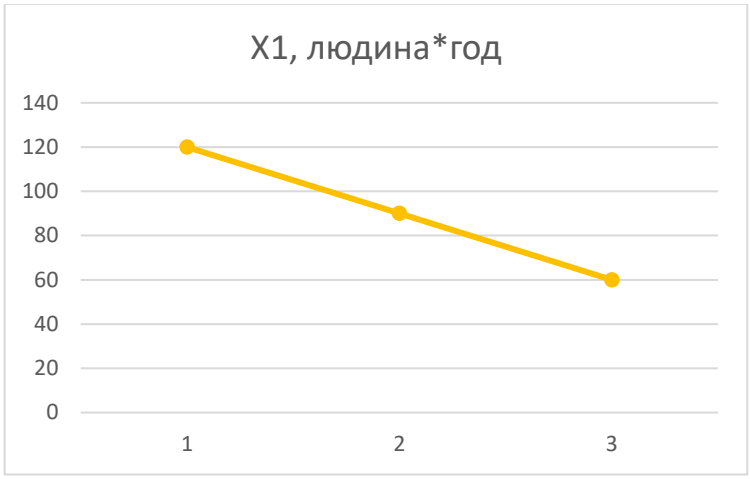


Рисунок 4.2 – X1, час розробки

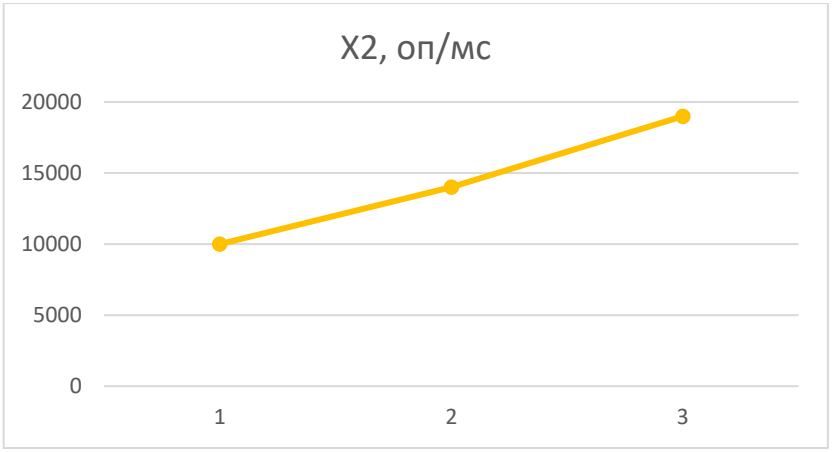


Рисунок 4.3 – X2, швидкодія мови програмування

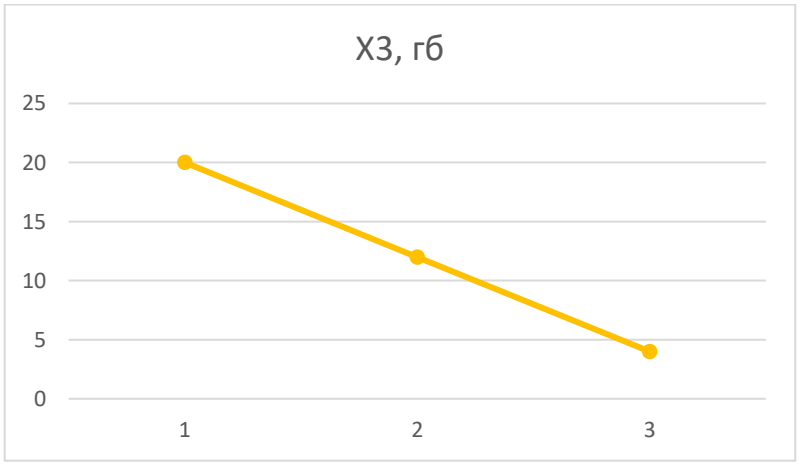


Рисунок 4.4 – X3, об’єм пам’яті для збереження даних

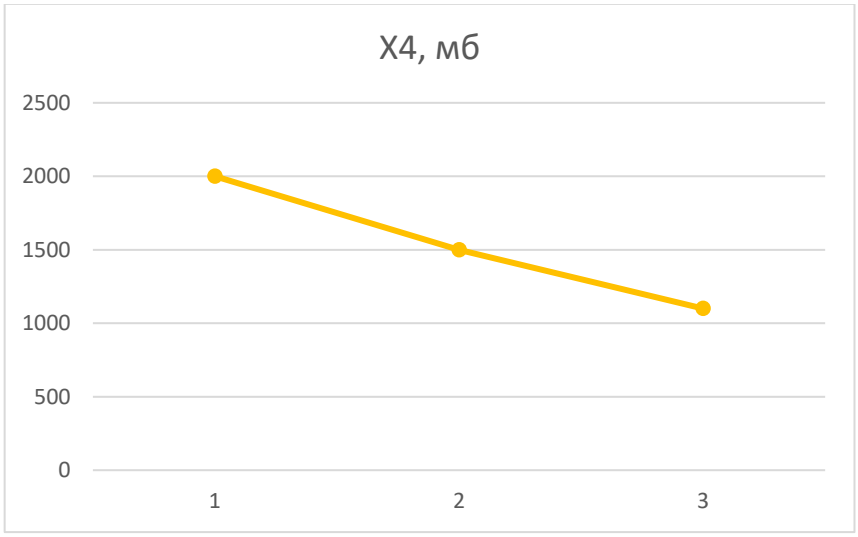


Рисунок 4.5 – X4, необхідний ресурс оперативної пам’яті

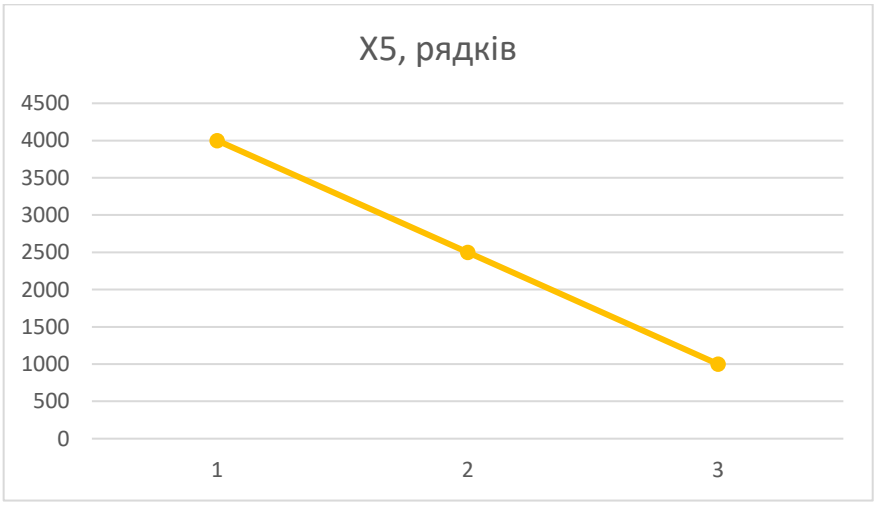


Рисунок 4.6 – X5, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень. Оцінку проводить експертна комісія із 7 людей. Значимість кожного

параметра визначається методом попарного порівняння. Визначення коефіцієнтів значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів;
– перевірку придатності експертних оцінок для подальшого використання;
– визначення оцінки попарного пріоритету параметрів;
– обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Час розробки	людина*год	4	5	5	5	3	4	5	31	10	100
X2	Швидкодія мови програмування	Оп/мс	5	3	2	4	4	3	3	24	3	9
X3	Об’єм пам’яті для збереження даних	Гб	2	1	1	2	1	2	1	10	-11	121

X4	Необхідний ресурс оперативної пам'яті	Мб	3	4	4	3	5	5	4	28	7	49
X5	Потенційний об'єм програмного коду	Кіл-ть рядків коду	1	2	3	1	2	1	2	12	-9	81
	Разом		15	15	15	15	15	15	15	105	0	360

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 105,$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 21$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T.$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 360.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 211}{7^2(4^3 - 4)} = 0{,}734 > W_k = 0{,}67.$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 та X2	<	>	>	>	<	>	>	>	1,5
X1 та X3	>	>	>	>	>	>	>	>	1,5
X1 та X4	>	>	>	>	<	<	>	>	1,5

X1 та X5	>	>	>	>	>	>	>	>	1,5
X2 та X3	>	>	>	>	>	>	>	>	1,5
X2 та X4	>	<	<	>	<	<	<	<	0,5
X2 та X5	>	>	<	>	>	>	>	>	1,5
X3 та X4	<	<	<	<	<	<	<	<	0,5
X3 та X5	>	<	<	>	<	>	<	<	0,5
X4 та X5	>	>	>	>	>	>	>	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}.$$

З отриманих числових оцінок переваги складемо матрицю $A=\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}$$

$$b_i = \sum_{i=1}^{N} a_{ij}$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'},$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j					Перша ітерація		Друга ітерація		Третя ітерація	
	X1	X2	X3	X4	X5	b_i	$K_{\text{ві}}$	b_i^2	$K_{\text{ві}}^2$	b_i^3	$K_{\text{ві}}^3$
X1	1,0	1,5	1,5	1,5	1,5	7	0,28	49	0,36	343	0,44
X2	0,5	1,0	1,5	0,5	1,5	5	0,2	25	0,19	125	0,16
X3	0,5	0,5	1,0	0,5	0,5	3	0,12	9	0,06	27	0,04
X4	0,5	1,5	1,5	1,0	1,5	6	0,24	36	0,26	216	0,28
X5	0,5	0,5	1,5	0,5	1,0	4	0,16	16	0,13	64	0,08
Всього:						25	1	135	1	775	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо. Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} ,$$

де n – кількість параметрів;

$K_{вi}$– коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	70	9	0,44	1,5
		X2	15000	7	0,16	1,12
F2	А	X3	10	5	0,04	0,2
		X4	1600	6	0,28	1,68
	Б	X3	18	3	0,04	0,12
		X4	1800	4	0,28	1,12
F3	А	X5	1500	8	0,08	0,64

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 1,5 + 1,12 + 0,2 + 1,68 + 0,64 = 5,14,$$

$$K_{K2} = 1,5 + 1,12 + 0,12 + 1,12 + 0,64 = 4,5.$$

Як видно з розрахунків, кращим є перший варіант(F1а- F2а-F3б), для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як

$$T_О = T_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М},$$

де $T_Р$ – трудомісткість розробки ПП;

$К_П$ – поправочний коефіцієнт;

$К_{СК}$ – коефіцієнт на складність вхідної інформації;

$К_М$ – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_Р$ =90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 90 \cdot 1{,}7 \cdot 0{,}8 = 122{,}4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_Р$ =27 людино-днів, $К_П$ =0.9, $К_{СК}$ = 1, $К_{СТ}$ =0.8:

$T_2 = 27 \cdot 0{,}9 \cdot 0{,}8 = 19{,}44$ людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$T_I = (122{,}4 + 19{,}44 + 4{,}8 + 19{,}44) \cdot 8 = 1328{,}64$ людино-годин.

$T_{II} = (122{,}4 + 19{,}44 + 6{,}91 + 19{,}44) \cdot 8 = 1345{,}52$ людино-годин.

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь розробник з окладом 25000 грн та аналітик в області даних з окладом 28000. Визначимо середню зарплату за годину за формулою:

$$С_ч = \frac{М}{T_m \cdot t} \text{ грн.},$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_ч = \frac{25000 + 28000}{2 \cdot 21 \cdot 8} = 157{,}73 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_{ч} \cdot Т_{i} \cdot К_{Д},$$

де $С_{ч}$– величина погодинної оплати праці програміста;

$Т_{i}$ – трудомісткість відповідного завдання;

$К_{Д}$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 157,73 \cdot 1328,64 \cdot 1,2 = 251479,66$ грн.

II. $С_{ЗП} = 157,73 \cdot 1345,52 \cdot 1,2 = 254674,64$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0,22 = 251479,66 \cdot 0,22 = 55325,52$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0,22 = 254674,64 \cdot 0,22 = 56028,42$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_{М}$)

Так як одна ЕОМ обслуговує одного розробника з окладом 25000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 25000 \cdot 0,2 = 60000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 60000 \cdot (1 + 0,2) = 72000 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0,22 = 72000 \cdot 0,22 = 15840 \text{грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 20000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1,15 \cdot 0,25 \cdot 20000 = 5750 \text{ грн.},$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1,15 \cdot 20000 \cdot 0,05 = 1150 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} =(Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0,9 =$$

$$= 1677,6 \text{ годин},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} =1677,6 \cdot 0,3 \cdot 0,9 \cdot 3,52 = 1594,39 \text{грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0,67 = 20000 \cdot 0,67 =13400 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_{А} + С_{Р} + С_{ЕЛ} + С_{Н},$$

$С_{ЕКС}$ = 72000 + 15840 + 5750 + 1150 + 1594,39+ 13400= 109734,39 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС}/ Т_{ЕФ} = 109734,39/1677,6 = 65,41 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_{М} = С_{М-Г} \cdot Т,$$

I. $С_{М} = 65,41 \cdot 1328,64 = 86906,34$грн.

II. $С_{М} = 65,41 \cdot 1345.52 = 88010,46$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_{Н} = С_{ЗП} \cdot 0,67,$$

I. $С_Н = 251479,66 \cdot 0,67 = 168491,37$ грн.

II. $С_Н = 254674,64 \cdot 0,67 = 170632$грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н,$$

I. $С_{ПП} = 251479,66 + 55325,52 + 86906,34 + 168491,37 = 562202,89$ грн.

II. $С_{ПП} = 254674,64 + 56028,42 + 88010,46 + 170632 = 569345,52$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР_j} = К_{Кj} / С_{Фj},$$

$$К_{ТЕР1} = 5,14/ 562202,89 = 0,91 \cdot 10^{-5},$$

$$К_{ТЕР2} = 4,5/ 569345,52 = 0,79 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}= 0,91 \cdot 10^{-5}$.

4.8 Висновки до розділу 4

В ході виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних функцій програмного продукту, визначено параметри, які характеризують програмний продукт, проведено експертне оцінювання параметрів та аналіз якості варіантів реалізації функцій. Також проведено економічний аналіз варіантів розробки – трудомісткість, витрати на заробітну плату та інші витрати. Можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 0,91 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування – Python;
- використання бібліотеки Tensorflow для навчання нейронних мереж;
- використання стандартного середовища розробки Jupyter Notebook;

Даний варіант виконання експертної системи надає можливість досягти оптимального балансу швидкодії, зручності програмного продукту і необхідних ресурсів на його створення.

ВИСНОВКИ

У зв’язку з автоматизацією та вдосконаленням виробничих процесів постає питання знаходження нових методів ефективної оброк візуальних даних. Робота над розпізнаванням патологій яблунь є, беззаперечно, дуже важливою для фермерських господарств, оскільки дозволяє покращувати якість та обсяг готової продукції та знижує ризик втрати частини саду через захворювання листяної маси. Під задачею розпізнавання розуміється аналіз та класифікація зображень до певної категорії хвороб або до деякої сукупності категорій.

Станом на теперішній час, існує багато підходів до вирішення даного завдання. Серед найвідоміших можна виокремити підходи, які базуються на використанні регресійних моделей, дерев рішень, застосування методів опорних векторів. Також надзвичайно потужним інструментом для вирішення задач комп’ютерного зору є згорткові нейронні мережі. Розуміючи математичні основи описаних підходів, можна ефективно використовувати їх для вирішення поставленого завдання. Проте, серед причин, які заважають розглянутим вище методам показувати бездоганну точність прогнозу, є: варіації візуальних симптомів одного захворювання у різних сортів, різні ракурси та освітленість, розміри зображень.

При розробці системи розпізнавання патологій було використано дані ресурсу Kaggle про стан листя яблучних дерев. Спроектовано модель навчання, яка дозволяє розділити цей процес на етапи, що покращує продуктивність роботи. Було вирішено питання розмаїття візуальних шумів, шляхом доповнення вхідного набору даних. В навчанні було порівняно структуру мереж EfficientNetB4 та DenseNet169, адаптовані до поставленої задачі. Було здійснено оцінювання якості розпізнавання за допомогою метрики F1-score. Більш точною виявилась модель з використанням EfficientNetB4, саме її було використано для подальшого дослідження, оскільки вона має оптимальну кількість параметрів, швидкодію та точність. Виконано налаштування порогів входження екземпляру

до кожного з класів. Проведено експеримент з новим тестовим набором, який показав досить високу якість роботи. Але побудовану модель можна покращити, шляхом розширення вхідного датасету, додавання більшої кількості аугментацій зображень.

Отримані результати навчання моделі було збережено для подальшого використання в готових рішення, які мають на меті зйомку в режимі реального часу. Такий підхід дозволить автоматизувати діагностику фермерських насаджень та покращить продуктивність виробництва.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. MIPT Machine Learning online specialization. Coursera online learning platform. URL: https://www.coursera.org/specializations/machine-learning-data-analysis (дата звернення: 03.05.2021).

2. Russell S. J., Norvig P. Artificial Intelligence: A Modern Approach, Third Edition. New Jersey: Prentice Hall, 2010. 1132 p.

3. W. Wriggers. Artificial Neural Networks and Pattern Recognition URL: https://biomachina.org/courses/imageproc/131.pdf (дата звернення: 05.05.2021).

4. N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A Convolutional Neural Network for Modelling Sentences. URL: http://arxiv.org/abs/1404.2188 (дата звернення: 26.04.2021).

5. Simonyan, Karen, and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015.

6. Charu C. Aggarwal. Neural Networks and Deep Learning: A Textbook. Springer, Cham. 2018. 497 p.

7. Кузнєцова Н.В. Аналіз фінансово-економічних даних. Частина 1: Дерева рішень (навчально методичні матеріали до курсу лекцій) . Київ: НТУУ «КПІ», 2017. 59 ст.

8. M. Tan and Q. V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. URL: http://arxiv.org/abs/1905.11946 (дата звернення: 10.05.2021).

9. G. Tsoumakas and I. Katakis. Multi-Label Classification: An Overview. *International Journal of Data Warehousing and Mining*. 2007. Vol. 3, P. 1–13.

10. J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. *Machine Learning*. 2011. Vol. 85. P. 333–359.

11. K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. URL: http://arxiv.org/abs/1512.03385 (дата звернення: 08.05.2021).

12. F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer. DenseNet: Implementing Efficient ConvNet Descriptor Pyramids. URL: http://arxiv.org/abs/1404.1869 (дата звернення: 08.05.2021).

13. R. Szeliski. Computer Vision: Algorithms and Applications. Springer Science & Business Media, 2010. 811 p.

14. S. A. J. Winder and M. Brown. Learning Local Image Descriptors. CVPR '07: IEEE Conference on Computer Vision and Pattern Recognition, Minneapolis. 2007, P. 1–8.

15. J. Howse. OpenCV Computer Vision with Python. Packt Publishing Ltd, 2013. 122 p.

16. J. E. Solem, Programming Computer Vision with Python: Tools and algorithms for analyzing images. O'Reilly Media, Inc., 2012. 264 p.

17. Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le. Self-training with Noisy Student improves ImageNet classification. URL: http://arxiv.org/abs/1911.04252 (дата звернення: 12.05.2021).

ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО ПРОДУКТУ

Модуль preprocessing.py

```
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
import numpy as np
import imagehash
import PIL
import os

class CFG0():
    threshold = .9
    img_size = 512
    seed = 42

root = 'train_images'
paths = os.listdir(root)
df = pd.read_csv('train.csv', index_col='image')

for path in tqdm(paths, total=len(paths)):
    image = tf.io.read_file(os.path.join(root, path))
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, [CFG0.img_size, CFG0.img_size])
    image = tf.cast(image, tf.uint8).numpy()
    plt.imsave(path, image)
hash_functions = [
    imagehash.average_hash,
```

```
    imagehash.phash,
    imagehash.dhash,
    imagehash.whash]

image_ids = []
hashes = []

paths = tf.io.gfile.glob('./*.jpg')

for path in tqdm(paths, total=len(paths)):
    image = PIL.Image.open(path)
    hashes.append(np.array([x(image).hash for x in hash_functions]).reshape(-1,))
    image_ids.append(path.split('/')[-1])

hashes = np.array(hashes)
image_ids = np.array(image_ids)

duplicate_ids = []

for i in tqdm(range(len(hashes)), total=len(hashes)):
    similarity = (hashes[i] == hashes).mean(axis=1)
    duplicate_ids.append(list(image_ids[similarity > CFG0.threshold]))

duplicates = [frozenset([x] + y) for x, y in zip(image_ids, duplicate_ids)]
duplicates = set([x for x in duplicates if len(x) > 1])
duplicates_by_kingofarmy = {
    frozenset(('8dbeda49894d522e.jpg', 'afbe5641896d522a.jpg')),
    frozenset(('af6292db1b611d98.jpg', 'a56292dadb618d95.jpg')),
    frozenset(('abf0b5a0df028b17.jpg', 'abf0b5819f028f0f.jpg')),
    frozenset(('e385830ecacd2d9e.jpg', 'c335971e8acd609e.jpg')),
```

```
    frozenset(('cebdc20f67838631.jpg', 'dfbdc047068b063d.jpg')),
    frozenset(('f392f11919991cea.jpg', 'f196f11a99d91ce0.jpg'))}

duplicates |= duplicates_by_kingofarmy

print(f'Found {len(duplicates)} duplicate pairs:')
for row in duplicates:
    print(', '.join(row))

with open('duplicates.csv', 'w') as file:
    for row in duplicates:
        file.write(','.join(row) + '\n')
for row in duplicates:
    figure, axes = plt.subplots(1, len(row), figsize=[5 * len(row), 5])
    for i, image_id in enumerate(row):
        image = plt.imread(os.path.join('train_images', image_id))
        axes[i].imshow(image)
        axes[i].set_title(f'{image_id} - {df.loc[image_id, "labels"]}')
        axes[i].axis('off')
    plt.show()
for file in tf.io.gfile.glob('./*.jpg'):
    os.remove(file)

from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import StratifiedKFold
import albumentations
import shutil

class CFG:
```

```
    root = 'train_images'
    classes = [
        'complex',
        'frog_eye_leaf_spot',
        'powdery_mildew',
        'rust',
        'scab',
        'healthy']
    strategy = tf.distribute.get_strategy()
    batch_size = 16
    img_size = 600
    folds = 5
    seed = 42
    subfolds = 16 # number of .tfrec files in each fold
    transform = True
    epochs = 5

df = pd.read_csv('train.csv', index_col='image')
init_len = len(df)

with open('duplicates.csv', 'r') as file:
    duplicates = [x.strip().split(',') for x in file.readlines()]

for row in duplicates:
    unique_labels = df.loc[row].drop_duplicates().values
    if len(unique_labels) == 1:
        df = df.drop(row[1:], axis=0)
    else:
        df = df.drop(row, axis=0)
```

```
print(f'Dropping {init_len - len(df)} duplicate samples.')
original_labels = df['labels'].values.copy()

df['labels'] = [x.split(' ') for x in df['labels']]
labels = MultiLabelBinarizer(classes=CFG.classes).fit_transform(df['labels'].values)
df = pd.DataFrame(columns=CFG.classes, data=labels, index=df.index)
df.to_csv('train.csv')
display(df.head())

kfold = StratifiedKFold(n_splits=CFG.folds, shuffle=True, random_state=CFG.seed)
fold = np.zeros((len(df),))

for i, (train_index, val_index) in enumerate(kfold.split(df.index, original_labels)):
    fold[val_index] = i

value_counts = lambda x: pd.Series.value_counts(x, normalize=True)

df_occurence = pd.DataFrame({
    'origin': df.apply(value_counts).loc[1],
    'fold_0': df[fold == 0].apply(value_counts).loc[1],
    'fold_1': df[fold == 1].apply(value_counts).loc[1],
    'fold_2': df[fold == 2].apply(value_counts).loc[1],
    'fold_3': df[fold == 3].apply(value_counts).loc[1],
    'fold_4': df[fold == 4].apply(value_counts).loc[1]})

bar = df_occurence.plot.barh(figsize=[15, 5], colormap='plasma')
folds = pd.DataFrame({
    'image': df.index,
    'fold': fold})
```

```
folds.to_csv('folds.csv', index=False)
if CFG.transform:
    transform = albumentations.Compose([
        albumentations.RandomResizedCrop(CFG.img_size, CFG.img_size, scale=(0.9, 1), p=1),
        albumentations.HorizontalFlip(p=0.5),
        albumentations.VerticalFlip(p=0.5),
        albumentations.ShiftScaleRotate(p=0.5),
        albumentations.HueSaturationValue(hue_shift_limit=10, sat_shift_limit=10, val_shift_limit=10, p=0.7),
        albumentations.RandomBrightnessContrast(brightness_limit=(-0.2,0.2), contrast_limit=(-0.2, 0.2), p=0.7),
        albumentations.CLAHE(clip_limit=(1,4), p=0.5),
        albumentations.OneOf([
            albumentations.OpticalDistortion(distort_limit=1.0),
            albumentations.GridDistortion(num_steps=5, distort_limit=1.),
            albumentations.ElasticTransform(alpha=3),
        ], p=0.2),
        albumentations.OneOf([
            albumentations.GaussNoise(var_limit=[10, 50]),
            albumentations.GaussianBlur(),
            albumentations.MotionBlur(),
            albumentations.MedianBlur(),
        ], p=0.2),
        albumentations.Resize(CFG.img_size, CFG.img_size),
        albumentations.OneOf([
            albumentations.JpegCompression(),
            albumentations.Downscale(scale_min=0.1, scale_max=0.15),
        ], p=0.2),
        albumentations.IAAPiecewiseAffine(p=0.2),
```

```
        albumentations.IAASharpen(p=0.2),
        albumentations.Cutout(max_h_size=int(CFG.img_size * 0.1),
max_w_size=int(CFG.img_size * 0.1), num_holes=5, p=0.5),
    ])
else:
    transform = None
figure, axes = plt.subplots(5, 5, figsize=[15, 15])
axes = axes.reshape(-1,)

if transform is None:
    for i in range(len(axes)):
        image = tf.io.read_file(os.path.join(CFG.root, df.index[i]))
        image = tf.image.decode_jpeg(image, channels=3)
        image = tf.image.resize(image, [CFG.img_size, CFG.img_size])
        image = tf.cast(image, tf.uint8)
        axes[i].imshow(image.numpy())
        axes[i].axis('off')

else:
    image = tf.io.read_file(os.path.join(CFG.root, df.index[CFG.seed]))
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, [CFG.img_size, CFG.img_size])
    image = tf.cast(image, tf.uint8)
    for i in range(len(axes)):
        axes[i].imshow(transform(image=image.numpy())['image'])
        axes[i].axis('off')
plt.show()

def _serialize_image(path, transform=None):
    image = tf.io.read_file(path)
```

```
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, [CFG.img_size, CFG.img_size])
    image = tf.cast(image, tf.uint8)

    if transform is not None:
        image = transform(image=image.numpy())['image']
    return tf.image.encode_jpeg(image).numpy()

def _serialize_sample(image, image_name, label):
    feature = {
        'image': tf.train.Feature(bytes_list=tf.train.BytesList(value=[image])),
        'image_name':
tf.train.Feature(bytes_list=tf.train.BytesList(value=[image_name])),
        'complex': tf.train.Feature(int64_list=tf.train.Int64List(value=[label[0]])),
        'frog_eye_leaf_spot':
tf.train.Feature(int64_list=tf.train.Int64List(value=[label[1]])),
        'powdery_mildew':
tf.train.Feature(int64_list=tf.train.Int64List(value=[label[2]])),
        'rust': tf.train.Feature(int64_list=tf.train.Int64List(value=[label[3]])),
        'scab': tf.train.Feature(int64_list=tf.train.Int64List(value=[label[4]])),
        'healthy': tf.train.Feature(int64_list=tf.train.Int64List(value=[label[5]]))}
    sample = tf.train.Example(features=tf.train.Features(feature=feature))
    return sample.SerializeToString()

def serialize_fold(fold, name, transform=None, bar=None):
    samples = []

    for image_name, labels in fold.iterrows():
        path = os.path.join(CFG.root, image_name)
        image = _serialize_image(path, transform=transform)
```

```
        samples.append(_serialize_sample(image, image_name.encode(), labels))

    with tf.io.TFRecordWriter(name + '.tfrec') as writer:
        [writer.write(x) for x in samples]

    if bar is not None:
        bar.update(1)
total = CFG.folds * CFG.subfolds if transform is None else CFG.folds * CFG.subfolds
* CFG.epochs

with tqdm(total=total) as bar:

    for i in range(CFG.folds):
        df_fold = df[fold == i]
        folder = f'fold_{i}'

        try:
            os.mkdir(folder)
        except FileExistsError:
            shutil.rmtree(folder)
            os.mkdir(folder)

        if transform is None:
            for k, subfold in enumerate(np.array_split(df_fold, CFG.subfolds)):
                name=os.path.join(folder, '%.2i-%.3i' % (k, len(subfold)))
                serialize_fold(subfold, name=name, bar=bar)
        else:
            for j in range(CFG.epochs):
                for k, subfold in enumerate(np.array_split(df_fold, CFG.subfolds)):
```

```
            name=os.path.join(folder, '%.2i-%.3i' % (j * CFG.subfolds + k, len(subfold)))
            serialize_fold(subfold, name=name, transform=transform, bar=bar)
```

Модуль Training.py

```
from kaggle_datasets import KaggleDatasets
from sklearn.model_selection import KFold
import efficientnet.tfkeras as efn
import matplotlib.pyplot as plt
import tensorflow_addons as tfa
import tensorflow as tf
import pandas as pd
import numpy as np
import os

try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
    print('Running on TPUv3-8')
except:
    tpu = None
    tf.keras.mixed_precision.set_global_policy('mixed_float16')
    strategy = tf.distribute.get_strategy()
    print('Running on GPU with mixed precision')

batch_size = 16 * strategy.num_replicas_in_sync
```

```
print('Number of replicas:', strategy.num_replicas_in_sync)
print('Batch size: %.i' % batch_size)
class CFG():

    strategy = strategy
    batch_size = batch_size

    img_size = 600
    classes = [
        'complex',
        'frog_eye_leaf_spot',
        'powdery_mildew',
        'rust',
        'scab']
    gcs_path_raw = 'kfold0'
    gcs_path_aug = ['kfold1 aug', 'kfold2 aug', 'kfold3 aug', 'kfold4 aug']
    seed = 42
    epochs = 100
    patience = [5, 2]
    factor = .1
    min_lr = 1e-8
    verbose = 2
    folds = 5
    used_folds = [0, 1, 2, 3, 4]

def count_data_items(filenames):
    return np.sum([int(x[:-6].split('-')[-1]) for x in filenames])

def decode_image(image_data):
    image = tf.image.decode_jpeg(image_data, channels=3)
```

```
    image = tf.reshape(image, [CFG.img_size, CFG.img_size, 3])
    image = tf.cast(image, tf.float32) / 255.
    return image

feature_map = {
    'image': tf.io.FixedLenFeature([], tf.string),
    'image_name': tf.io.FixedLenFeature([], tf.string),
    'complex': tf.io.FixedLenFeature([], tf.int64),
    'frog_eye_leaf_spot': tf.io.FixedLenFeature([], tf.int64),
    'powdery_mildew': tf.io.FixedLenFeature([], tf.int64),
    'rust': tf.io.FixedLenFeature([], tf.int64),
    'scab': tf.io.FixedLenFeature([], tf.int64),
    'healthy': tf.io.FixedLenFeature([], tf.int64)}

def read_tfrecord(example, labeled=True):
    example = tf.io.parse_single_example(example, feature_map)
    image = decode_image(example['image'])
    if labeled:
        label = [tf.cast(example[x], tf.float32) for x in CFG.classes]
    else:
        label = example['image_name']
    return image, label

def get_dataset(filenames, labeled=True, ordered=True, shuffled=False,
                repeated=False, cached=False, distributed=True):
    auto = tf.data.experimental.AUTOTUNE
    dataset = tf.data.TFRecordDataset(filenames, num_parallel_reads=auto)
    if not ordered:
        ignore_order = tf.data.Options()
        ignore_order.experimental_deterministic = False
```

```
        dataset = dataset.with_options(ignore_order)
    dataset = dataset.map(
        lambda x: read_tfrecord(x, labeled=labeled),
        num_parallel_calls=auto)
    if shuffled:
        dataset = dataset.shuffle(2048, seed=CFG.seed)
    if repeated:
        dataset = dataset.repeat()
    dataset = dataset.batch(CFG.batch_size)
    if cached:
        dataset = dataset.cache()
    dataset = dataset.prefetch(auto)
    if distributed:
        dataset = CFG.strategy.experimental_distribute_dataset(dataset)
    return dataset

def get_model():
    model = tf.keras.models.Sequential(name='EfficientNetB4')
    model.add(efn.EfficientNetB4(
        include_top=False,
        input_shape=(CFG.img_size, CFG.img_size, 3),
        weights='noisy-student',
        pooling='avg'))

    model.add(tf.keras.layers.Dense(len(CFG.classes),
        kernel_initializer=tf.keras.initializers.RandomUniform(seed=CFG.seed),
        bias_initializer=tf.keras.initializers.Zeros(), name='dense_top'))
    model.add(tf.keras.layers.Activation('sigmoid', dtype='float32'))

    return model
```

```
filenames = tf.io.gfile.glob(os.path.join(CFG.gcs_path_aug[0], 'fold_0/*.tfrec'))[:1]

dataset = get_dataset(filenames, ordered=False, distributed=False)

plt.figure(figsize=[15, 15])

for i, sample in enumerate(dataset.unbatch().take(25).as_numpy_iterator()):
    plt.subplot(5, 5, i + 1)
    plt.imshow(sample[0])
    plt.axis('off')

plt.show()
model = get_model()
model.summary()
histories = []
scores = []
image_names = np.empty((0,))
predicts = np.empty((0, len(CFG.classes)))

callbacks = [
    tf.keras.callbacks.EarlyStopping(
        monitor='val_f1_score', mode='max',
        patience=CFG.patience[0], restore_best_weights=True),
    tf.keras.callbacks.ReduceLROnPlateau(
        monitor='val_f1_score', mode='max',
        patience=CFG.patience[1], min_lr=CFG.min_lr, verbose=2)]

kfold = KFold(n_splits=CFG.folds, shuffle=True, random_state=CFG.seed)
folds = ['fold_0', 'fold_1', 'fold_2', 'fold_3', 'fold_4']
```

```
for i, (train_index, val_index) in enumerate(kfold.split(folds)):

    if i in CFG.used_folds:
        print('=' * 74)
        print(f'Fold {i}')
        print('=' * 74)

        if tpu is not None:
            tf.tpu.experimental.initialize_tpu_system(tpu)

        with CFG.strategy.scope():
            model = get_model()

            model.compile(
                loss=tf.keras.losses.BinaryCrossentropy(),
                optimizer='adam',
                metrics=[
                    tf.keras.metrics.BinaryAccuracy(name='acc'),
                     tfa.metrics.F1Score(
                        num_classes=len(CFG.classes),
                        average='macro')])

        train_filenames = []
        for j in train_index:
            train_filenames += tf.io.gfile.glob(os.path.join(CFG.gcs_path_aug[0], folds[j],
'*.tfrec'))
            train_filenames += tf.io.gfile.glob(os.path.join(CFG.gcs_path_aug[1], folds[j],
'*.tfrec'))
```

```
        train_filenames += tf.io.gfile.glob(os.path.join(CFG.gcs_path_aug[2], folds[j], '*.tfrec'))
        train_filenames += tf.io.gfile.glob(os.path.join(CFG.gcs_path_aug[3], folds[j], '*.tfrec'))
    np.random.shuffle(train_filenames)

    val_filenames = []
    for j in val_index:
        val_filenames += tf.io.gfile.glob(os.path.join(CFG.gcs_path_raw, folds[j], '*.tfrec'))

    train_dataset = get_dataset(
        train_filenames,
         ordered=False, shuffled=True, repeated=True)

    val_dataset = get_dataset(
        val_filenames,
        cached=True)

    steps_per_epoch = count_data_items(train_filenames) // (20 * CFG.batch_size)
    validation_steps = count_data_items(val_filenames) // CFG.batch_size

    history = model.fit(
        train_dataset,
        steps_per_epoch=steps_per_epoch,
        validation_data=val_dataset,
        validation_steps=validation_steps,
        callbacks=callbacks,
        epochs=CFG.epochs,
        verbose=CFG.verbose).history
```

```
        size = count_data_items(val_filenames)
        steps = size // CFG.batch_size + 1

        val_dataset = get_dataset(val_filenames, labeled=False, distributed=False)
        val_predicts = model.predict(
            val_dataset.map(lambda x, y: x),
            steps=steps,
            verbose=CFG.verbose)[:size]
        val_image_names = [x.decode() for x in val_dataset.map(lambda x, y: y).unbatch().take(size).as_numpy_iterator()]

        image_names = np.concatenate((image_names, val_image_names))
        predicts = np.concatenate((predicts, val_predicts))

        model.save_weights(f'model_{i}.h5')
        histories.append(pd.DataFrame(history))
        scores.append(histories[-1]['val_f1_score'].max())

    else:
        pass
scores_df = pd.DataFrame({
    'fold': np.arange(len(scores)),
    'f1': np.round(scores, 4)})

with pd.option_context('display.max_rows', None):
    display(scores_df)

print('CV %.4f' % scores_df['f1'].mean())
figure, axes = plt.subplots(1, 5, figsize=[20, 5])
```

```
for i in range(CFG.folds):

    try:
        axes[i].plot(histories[i].loc[:, 'f1_score'], label='train')
        axes[i].plot(histories[i].loc[:, 'val_f1_score'], label='val')
        axes[i].legend()
    except IndexError:
        pass

    axes[i].set_title(f'fold {i}')
    axes[i].set_xlabel('epochs')

plt.show()
predicts_df = pd.DataFrame(
    columns=CFG.classes,
    data=predicts,
    index=pd.Index(data=image_names, name='image'))

predicts_df.to_csv('valid_predicts.csv')
display(predicts_df.head())
```

Модуль Training_DN.py

```
from sklearn.preprocessing import MultiLabelBinarizer
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
import tensorflow_addons as tfa
import tensorflow as tf
import pandas as pd
```

```
import numpy as np
import random
import os
from tensorflow.keras.applications.densenet import DenseNet169
from keras.layers.pooling import GlobalAveragePooling2D
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.models import Sequential, load_model, Model
try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
    print('Running on TPUv3-8')
except:
    tpu = None
    tf.keras.mixed_precision.set_global_policy('mixed_float16')
    strategy = tf.distribute.get_strategy()
    print('Running on GPU with mixed precision')

batch_size = 16 * strategy.num_replicas_in_sync

print('Number of replicas:', strategy.num_replicas_in_sync)
print('Batch size: %.i' % batch_size)
class CFG():

    strategy = strategy
    batch_size = batch_size

    img_size = 600
    classes = [
```

```
        'complex',
        'frog_eye_leaf_spot',
        'powdery_mildew',
        'rust',
        'scab']
    gcs_path_raw = 'kfold0'
    gcs_path_aug = ['kfold1 aug', 'kfold2 aug', 'kfold3 aug', 'kfold4 aug']
    seed = 42
    epochs = 100
    patience = [5, 2]
    factor = .1
    min_lr = 1e-8
    verbose = 2
    folds = 5
    used_folds = [0, 1, 2, 3, 4]

def count_data_items(filenames):
    return np.sum([int(x[:-6].split('-')[-1]) for x in filenames])

def decode_image(image_data):
    image = tf.image.decode_jpeg(image_data, channels=3)
    image = tf.reshape(image, [CFG.img_size, CFG.img_size, 3])
    image = tf.cast(image, tf.float32) / 255.
    return image

feature_map = {
    'image': tf.io.FixedLenFeature([], tf.string),
    'image_name': tf.io.FixedLenFeature([], tf.string),
    'complex': tf.io.FixedLenFeature([], tf.int64),
    'frog_eye_leaf_spot': tf.io.FixedLenFeature([], tf.int64),
```

```
    'powdery_mildew': tf.io.FixedLenFeature([], tf.int64),
    'rust': tf.io.FixedLenFeature([], tf.int64),
    'scab': tf.io.FixedLenFeature([], tf.int64),
    'healthy': tf.io.FixedLenFeature([], tf.int64)}

def read_tfrecord(example, labeled=True):
    example = tf.io.parse_single_example(example, feature_map)
    image = decode_image(example['image'])
    if labeled:
        label = [tf.cast(example[x], tf.float32) for x in CFG.classes]
    else:
        label = example['image_name']
    return image, label

def get_dataset(filenames, labeled=True, ordered=True, shuffled=False,
                repeated=False, cached=False, distributed=True):
    auto = tf.data.experimental.AUTOTUNE
    dataset = tf.data.TFRecordDataset(filenames, num_parallel_reads=auto)
    if not ordered:
        ignore_order = tf.data.Options()
        ignore_order.experimental_deterministic = False
        dataset = dataset.with_options(ignore_order)
    dataset = dataset.map(
        lambda x: read_tfrecord(x, labeled=labeled),
        num_parallel_calls=auto)
    if shuffled:
        dataset = dataset.shuffle(2048, seed=CFG.seed)
    if repeated:
        dataset = dataset.repeat()
    dataset = dataset.batch(CFG.batch_size)
```

```
    if cached:
        dataset = dataset.cache()
    dataset = dataset.prefetch(auto)
    if distributed:
        dataset = CFG.strategy.experimental_distribute_dataset(dataset)
    return dataset

def get_model():
    base_model = DenseNet169(include_top=False, weights='imagenet', input_shape=(CFG.img_size, CFG.img_size, 3))
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.2)(x)
    x = Dense(64, activation='relu')(x)
    prediction = Dense(5, activation='sigmoid')(x)

    model = Model(inputs=base_model.input, outputs=prediction)
    return model

filenames = tf.io.gfile.glob(os.path.join(CFG.gcs_path_aug[0], 'fold_0/*.tfrec'))[:1]

dataset = get_dataset(filenames, ordered=False, distributed=False)

plt.figure(figsize=[15, 15])

for i, sample in enumerate(dataset.unbatch().take(25).as_numpy_iterator()):
    plt.subplot(5, 5, i + 1)
    plt.imshow(sample[0])
    plt.axis('off')
```

```
plt.show()
model = get_model()
model.summary()
histories = []
scores = []
image_names = np.empty((0,))
predicts = np.empty((0, len(CFG.classes)))

callbacks = [
    tf.keras.callbacks.EarlyStopping(
        monitor='val_f1_score', mode='max',
        patience=CFG.patience[0], restore_best_weights=True),
    tf.keras.callbacks.ReduceLROnPlateau(
        monitor='val_f1_score', mode='max',
        patience=CFG.patience[1], min_lr=CFG.min_lr, verbose=2)]

kfold = KFold(n_splits=CFG.folds, shuffle=True, random_state=CFG.seed)
folds = ['fold_0', 'fold_1', 'fold_2', 'fold_3', 'fold_4']


for i, (train_index, val_index) in enumerate(kfold.split(folds)):

    if i in CFG.used_folds:
        print('=' * 74)
        print(f'Fold {i}')
        print('=' * 74)

        if tpu is not None:
            tf.tpu.experimental.initialize_tpu_system(tpu)
```

```
    with CFG.strategy.scope():
        model = get_model()

        model.compile(
            loss=tf.keras.losses.BinaryCrossentropy(),
            optimizer='adam',
            metrics=[
                tf.keras.metrics.BinaryAccuracy(name='acc'),
                 tfa.metrics.F1Score(
                    num_classes=len(CFG.classes),
                    average='macro')])

    train_filenames = []
    for j in train_index:
        train_filenames += tf.io.gfile.glob(os.path.join(CFG.gcs_path_aug[0], folds[j],
'*.tfrec'))
        train_filenames += tf.io.gfile.glob(os.path.join(CFG.gcs_path_aug[1], folds[j],
'*.tfrec'))
        train_filenames += tf.io.gfile.glob(os.path.join(CFG.gcs_path_aug[2], folds[j],
'*.tfrec'))
        train_filenames += tf.io.gfile.glob(os.path.join(CFG.gcs_path_aug[3], folds[j],
'*.tfrec'))
    np.random.shuffle(train_filenames)

    val_filenames = []
    for j in val_index:
        val_filenames += tf.io.gfile.glob(os.path.join(CFG.gcs_path_raw, folds[j],
'*.tfrec'))
```

```
    train_dataset = get_dataset(
        train_filenames,
         ordered=False, shuffled=True, repeated=True)

    val_dataset = get_dataset(
        val_filenames,
        cached=True)

    steps_per_epoch = count_data_items(train_filenames) // (20 * CFG.batch_size)
    validation_steps = count_data_items(val_filenames) // CFG.batch_size

    history = model.fit(
        train_dataset,
        steps_per_epoch=steps_per_epoch,
        validation_data=val_dataset,
        validation_steps=validation_steps,
        callbacks=callbacks,
        epochs=CFG.epochs,
        verbose=CFG.verbose).history

    size = count_data_items(val_filenames)
    steps = size // CFG.batch_size + 1

    val_dataset = get_dataset(val_filenames, labeled=False, distributed=False)
    val_predicts = model.predict(
        val_dataset.map(lambda x, y: x),
        steps=steps,
        verbose=CFG.verbose)[:size]
    val_image_names = [x.decode() for x in val_dataset.map(lambda x, y:
y).unbatch().take(size).as_numpy_iterator()]
```

```
            image_names = np.concatenate((image_names, val_image_names))
            predicts = np.concatenate((predicts, val_predicts))

            model.save_weights(f'model_{i}.h5')
            histories.append(pd.DataFrame(history))
            scores.append(histories[-1]['val_f1_score'].max())

        else:
            pass
scores_df = pd.DataFrame({
    'fold': np.arange(len(scores)),
    'f1': np.round(scores, 4)})

with pd.option_context('display.max_rows', None):
    display(scores_df)

print('CV %.4f' % scores_df['f1'].mean())
figure, axes = plt.subplots(1, 5, figsize=[20, 5])

for i in range(CFG.folds):

    try:
        axes[i].plot(histories[i].loc[:, 'f1_score'], label='train')
        axes[i].plot(histories[i].loc[:, 'val_f1_score'], label='val')
        axes[i].legend()
    except IndexError:
        pass

    axes[i].set_title(f'fold {i}')
```

```
    axes[i].set_xlabel('epochs')

plt.show()
predicts_df = pd.DataFrame(
    columns=CFG.classes,
    data=predicts,
    index=pd.Index(data=image_names, name='image'))

predicts_df.to_csv('valid_predicts_dn.csv')
display(predicts_df.head())
```

Модуль Test.py

```
from sklearn.preprocessing import MultiLabelBinarizer
from tqdm.notebook import tqdm
import efficientnet.tfkeras as efn
import matplotlib.pyplot as plt
import tensorflow_addons as tfa
import tensorflow as tf
import pandas as pd
import numpy as np
import random
import os

class CFG:
    strategy = tf.distribute.get_strategy()
    batch_size = 16 * strategy.num_replicas_in_sync
    img_size = 600

    classes = np.array([
```

```
        'complex',
        'frog_eye_leaf_spot',
        'powdery_mildew',
        'rust',
        'scab'])
    root = 'test_images'
    seed = 42
    tta_steps = 0

def decode_image(image_data):
    image = tf.image.decode_jpeg(image_data, channels=3)
    image = tf.reshape(image, [CFG.img_size, CFG.img_size, 3])
    image = tf.cast(image, tf.float32) / 255.
    return image
def data_augment(image, label):
    image = tf.image.random_flip_left_right(image, seed=CFG.seed)
    image = tf.image.random_flip_up_down(image, seed=CFG.seed)
     k = tf.tf.random.uniform([], minval=0, maxval=4, dtype=tf.int64, seed=CFG.seed)
    image = tf.image.rot90(image, k=k)

    image = tf.image.random_hue(image, .1, seed=CFG.seed)
    image = tf.image.random_saturation(image, .8, 1.2, seed=CFG.seed)
    image = tf.image.random_contrast(image, .8, 1.2, seed=CFG.seed)
    image = tf.image.random_brightness(image, .1, seed=CFG.seed)

    return image, label

feature_map = {
    'image': tf.io.FixedLenFeature([], tf.string),
    'image_name': tf.io.FixedLenFeature([], tf.string)}
```

```
def read_tfrecord(example):
    example = tf.io.parse_single_example(example, feature_map)
    image = decode_image(example['image'])
    label = example['image_name']
    return image, label

def get_dataset(filenames, ordered=True, shuffled=False, repeated=False,
                augmented=False, cached=False, distributed=False):
    auto = tf.data.experimental.AUTOTUNE
    dataset = tf.data.TFRecordDataset(filenames, num_parallel_reads=auto)
    if not ordered:
        ignore_order = tf.data.Options()
        ignore_order.experimental_deterministic = False
        dataset = dataset.with_options(ignore_order)
    dataset = dataset.map(read_tfrecord, num_parallel_calls=auto)
    if shuffled:
        dataset = dataset.shuffle(2048, seed=CFG.seed)
    if repeated:
        dataset = dataset.repeat()
    dataset = dataset.batch(CFG.batch_size)
    if augmented:
        dataset = dataset.map(data_augment, num_parallel_calls=auto)
    if cached:
        dataset = dataset.cache()
    dataset = dataset.prefetch(auto)
    if distributed:
        dataset = CFG.strategy.experimental_distribute_dataset(dataset)
    return dataset
```

```
def get_model():
    model = tf.keras.models.Sequential(name='EfficientNetB4')

    model.add(efn.EfficientNetB4(
        include_top=False,
        input_shape=(CFG.img_size, CFG.img_size, 3),
        weights=None,
        pooling='avg'))

    model.add(tf.keras.layers.Dense(len(CFG.classes),
        kernel_initializer=tf.keras.initializers.RandomUniform(seed=CFG.seed),
        bias_initializer=tf.keras.initializers.Zeros(), name='dense_top'))
    model.add(tf.keras.layers.Activation('sigmoid', dtype='float32'))

    return model

def _serialize_image(path):
    image = tf.io.read_file(path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, [CFG.img_size, CFG.img_size])
    image = tf.cast(image, tf.uint8)
    return tf.image.encode_jpeg(image).numpy()

def _serialize_sample(image, name):
    feature = {
        'image': tf.train.Feature(bytes_list=tf.train.BytesList(value=[image])),
        'image_name': tf.train.Feature(bytes_list=tf.train.BytesList(value=[name]))}
    sample = tf.train.Example(features=tf.train.Features(feature=feature))
    return sample.SerializeToString()
```

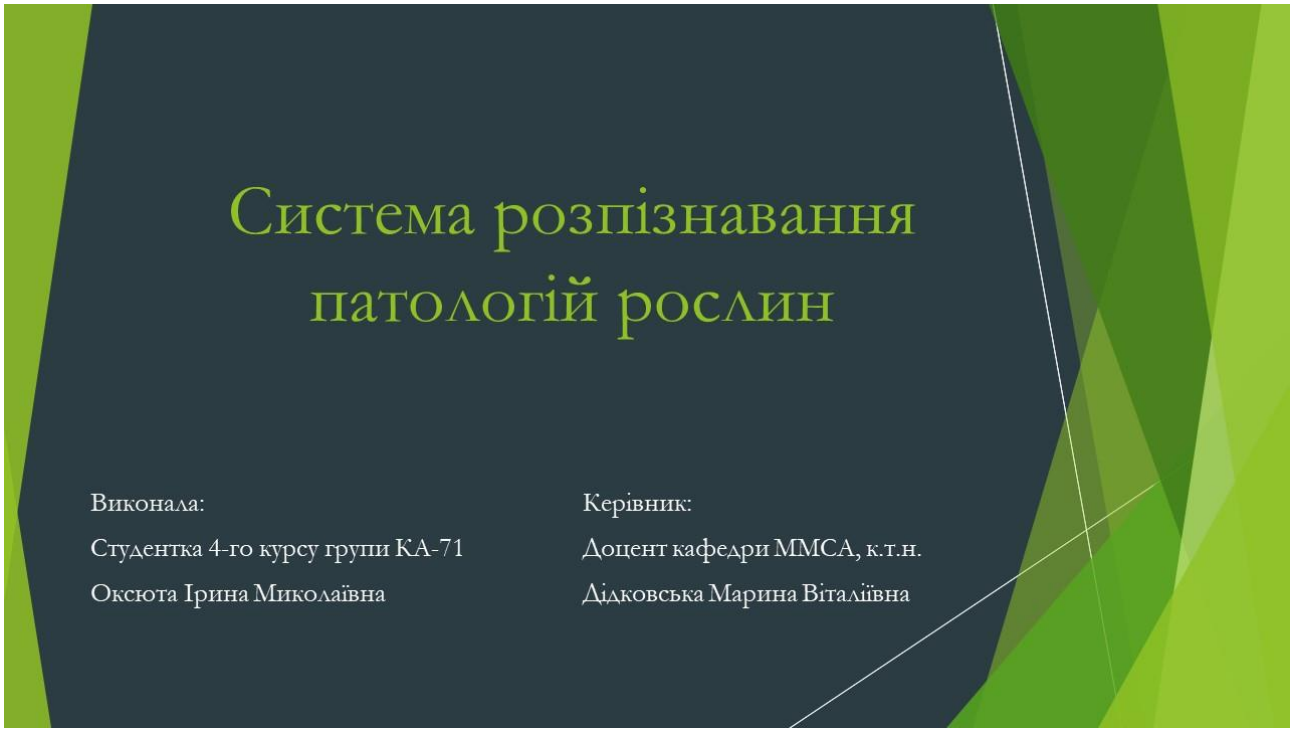
```
def serialize_test():
    samples = []
    for path in os.listdir(CFG.root):
        image = _serialize_image(os.path.join(CFG.root, path))
        name = path.encode()
        samples.append(_serialize_sample(image, name))

    with tf.io.TFRecordWriter('test.tfrec') as writer:
        [writer.write(x) for x in tqdm(samples, total=len(samples))]
paths = os.listdir(CFG.root)[:3]

figure, axes = plt.subplots(1, 3, figsize=[20, 10])

for i, path in enumerate(paths):
    image = plt.imread(os.path.join(CFG.root, path))

    axes[i].imshow(image)
    axes[i].axis('off')

plt.show()
size = len(os.listdir(CFG.root))
filenames = tf.io.gfile.glob('*.tfrec')

if CFG.tta_steps > 0:
    dataset = get_dataset(filenames, repeated=True, augmented=False)
else:
    dataset = get_dataset(filenames)

predicts = np.zeros((size, len(CFG.classes)))
paths = tf.io.gfile.glob('models/*.h5')
```

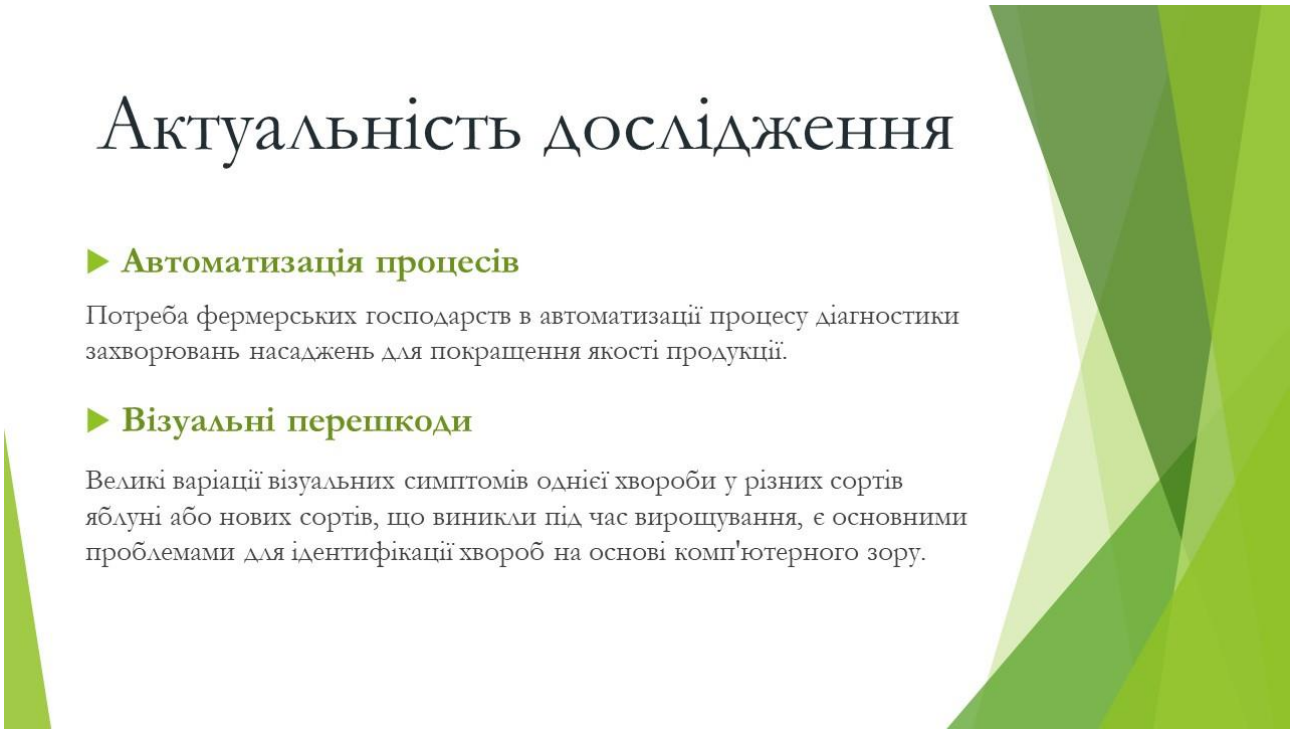
```
for path in tqdm(paths, total=len(paths)):

    with CFG.strategy.scope():
        model = get_model()
        model.load_weights(path)

    if CFG.tta_steps > 0:
        steps = CFG.tta_steps * (size / CFG.batch_size + 1)

        predict = model.predict(dataset, steps=steps)[:size * CFG.tta_steps] / len(paths)
        predicts += np.mean(
            predict.reshape(size, CFG.tta_steps, len(CFG.classes), order='F'), axis=1)

    else:
        predicts += model.predict(dataset) / len(paths)
df_true = pd.read_csv(
    'train10.csv', index_col='image').drop('healthy', axis=1)
df_pred = pd.read_csv(
    'valid_predicts.csv', index_col='image')

df_true = df_true.reindex(df_pred.index)
y_true = df_true.values
y_pred = df_pred.values

thresholds = np.arange(.01, 1., .01)
scores = []

for threshold in thresholds:
    metric = tfa.metrics.F1Score(
```

```
        num_classes=len(CFG.classes),
        average=None,
        threshold=threshold)
    metric.update_state(y_true, y_pred)
    scores.append(metric.result().numpy())

df = pd.DataFrame(columns=CFG.classes, data=scores, index=pd.Index(thresholds, name='threshold'))

thresholds = []
scores = []

for x in CFG.classes:
    thresholds.append(df[x].idxmax())
    scores.append(df[x].max())
    print(f'{x}: {df.loc[.5, x]:.4f} >>> {df.loc[thresholds[-1], x]:.4f} ({thresholds[-1]:.2f})')

print(f'\nmean score: {df.loc[.5].mean():.4f} >>> {np.mean(scores):.4f}')

predicts_df = pd.DataFrame(
    columns=CFG.classes,
    data=predicts,
    index=pd.Index(data=os.listdir(CFG.root), name='image'))

predicts_df.to_csv('predicts.csv')
display(predicts_df.head())
pred=predicts.copy()
for i in range(len(pred)):
    pred[i] = pred[i] > thresholds
```

```
pred = pred.astype('bool')
labels = []

for i in range(len(pred)):
    labels.append(' '.join(CFG.classes[pred[i]]))

labels = ['healthy' if ('healthy' in x or x == '') else x for x in labels]

df = pd.DataFrame({
    'image': os.listdir(CFG.root),
    'labels': labels})

df.to_csv('submission.csv', index=False)
display(df.head())
```

ДОДАТОК Б ПРЕЗЕНТАЦІЯ





Постановка задачі

- Предмет дослідження:

Прогнозування категорій захворювань рослин на основі моделей машинного навчання.

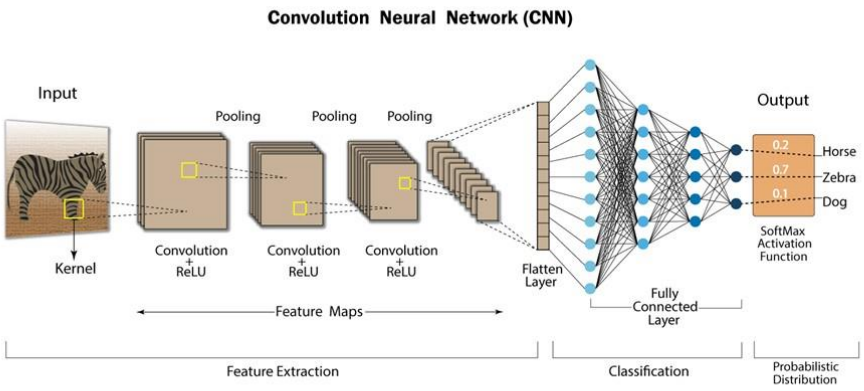
- Об'єкт дослідження:

Набір з 18632 високоякісних RGB зображень позакореневих захворювань яблунь, розмічених експертами на окремі категорії хвороб.

- Мета дослідження:

Розробка системи розпізнавання для точної класифікації окремих зображень листя до певної категорії хвороби, а також для виявлення окремого захворювання із множинних симптомів на одному зображенні

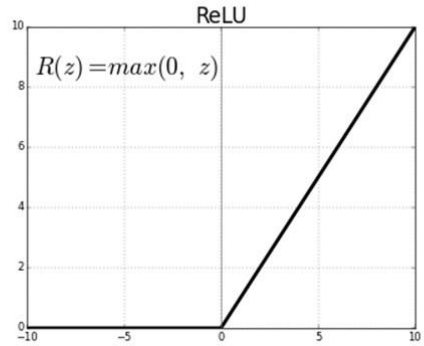
1) **Аналіз існуючих підходів до задач комп'ютерного зору**
2) **Передобробка вихідних даних та вирішення проблеми візуальних варіацій**
3) **Побудова моделей розпізнавання сукупності міток патологій для зображень листків яблунь**
4) **Оцінка роботи моделі на тестовому наборі та аналіз отриманих результатів**

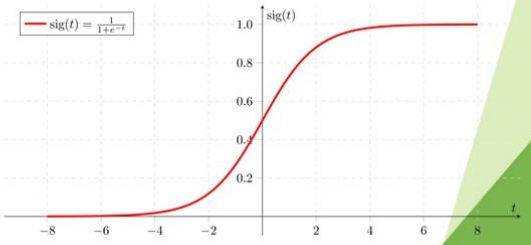


- Один з методів глибинного навчання
- Зменшує кількість параметрів, не втрачаючи якість
- Найчастіше використовується для задач розпізнавання образів

Функції активації

- Для поставленої задачі в згорткових шарах доцільно використовувати функцію активації ReLU. Вона є менш ресурсозатратною в порівнянні з tanh та сигмоїдою та дозволяє відсікати непотрібні деталі та проріджувати ваги.
- Натомість в повнозв'язному шарі в умовах задачі Multi-label classification використовують функцію активації sigmoid оскільки вона дає ймовірнісну оцінку, це в свою чергу дозволяє знайти ймовірність приналежності кожному окремому класу.





Метрики

Macro Averaged metrics

$$A^j_{Macro} = \frac{\sum_{i=1}^{n}[y_j^{(i)} \wedge \hat{y}_j^{(i)}]}{\sum_{i=1}^{n}[y_j^{(i)} \vee \hat{y}_j^{(i)}]}, \qquad Accuracy_{Macro} = \frac{1}{k}\sum_{j=1}^{k} A^j_{Macro}$$

$$P^j_{Macro} = \frac{\sum_{i=1}^{n}[y_j^{(i)} \wedge \hat{y}_j^{(i)}]}{\sum_{i=1}^{n}[\hat{y}_j^{(i)}]}, \qquad Precision_{Macro} = \frac{1}{k}\sum_{j=1}^{k} P^j_{Macro}$$

$$R^j_{Macro} = \frac{\sum_{i=1}^{n}[y_j^{(i)} \wedge \hat{y}_j^{(i)}]}{\sum_{i=1}^{n}[y_j^{(i)}]}, \qquad Recall_{Macro} = \frac{1}{k}\sum_{j=1}^{k} R^j_{Macro}$$

$$F1_{Macro} = \frac{2 * precision_{Macro} * recall_{Macro}}{(precision_{Macro} + recall_{Macro})}$$

Micro Averaged metrics

$$Accuracy_{Micro} = \frac{\sum_{j=1}^{k}\sum_{i=1}^{n}[y_j^{(i)} \wedge \hat{y}_j^{(i)}]}{\sum_{j=1}^{k}\sum_{i=1}^{n}[y_j^{(i)} \vee \hat{y}_j^{(i)}]}$$

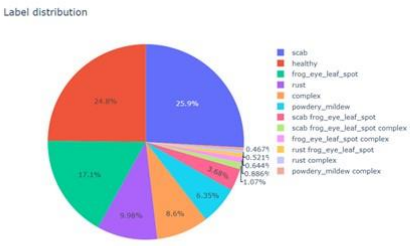
$$Precision_{Micro} = \frac{\sum_{j=1}^{k}\sum_{i=1}^{n}[y_j^{(i)} \wedge \hat{y}_j^{(i)}]}{\sum_{j=1}^{k}\sum_{i=1}^{n}\hat{y}_j^{(i)}}$$

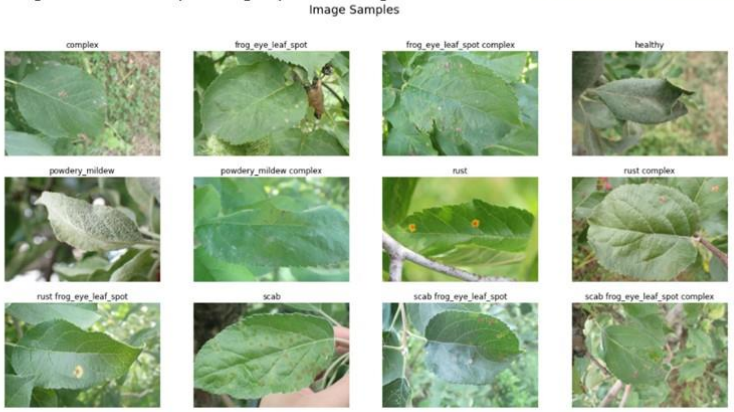
$$Recall_{Micro} = \frac{\sum_{j=1}^{k}\sum_{i=1}^{n}[y_j^{(i)} \wedge \hat{y}_j^{(i)}]}{\sum_{j=1}^{k}\sum_{i=1}^{n} y_j^{(i)}}$$

$$F1_{Micro} = \frac{2 * precision_{Micro} * recall_{Micro}}{(precision_{Micro} + recall_{Micro})}$$

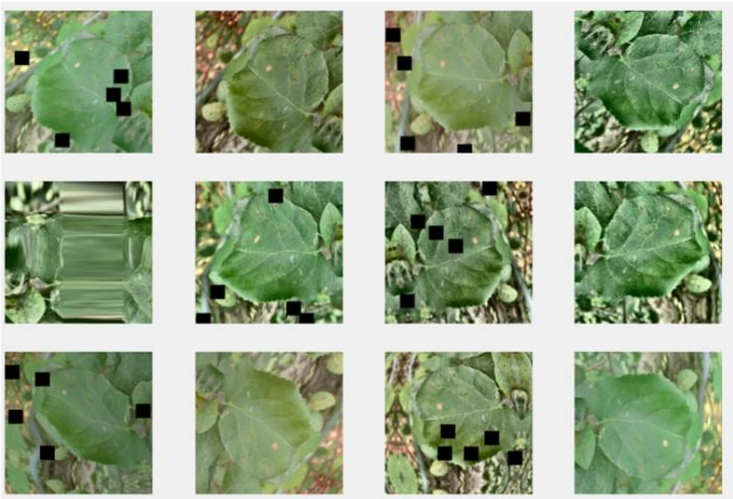
Вхідні дані

Набір даних сформований для змагання «Plant Pathology 2021 FGVC8» на ресурсі http://kaggle.com. Навчальна вибірка містить більш ніж 18 тисяч високоякісних зображень, що анотовані на окремі категорії хвороб відповідними експертами. Цей датасет відображає реальні польові сценарії, представляючи неоднорідні фони зображень листків, зроблених на різних стадіях зрілості та в різний час доби за різних налаштувань фокусної камери.



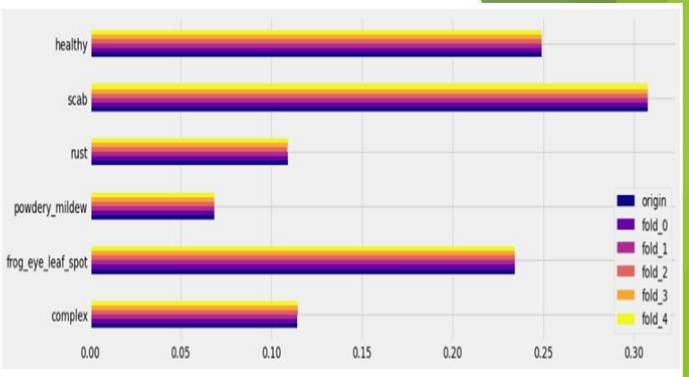


Аугментації зображень

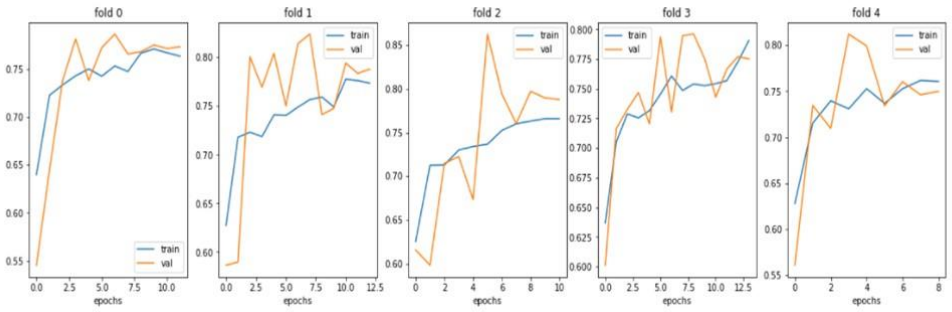


Запропоновані моделі

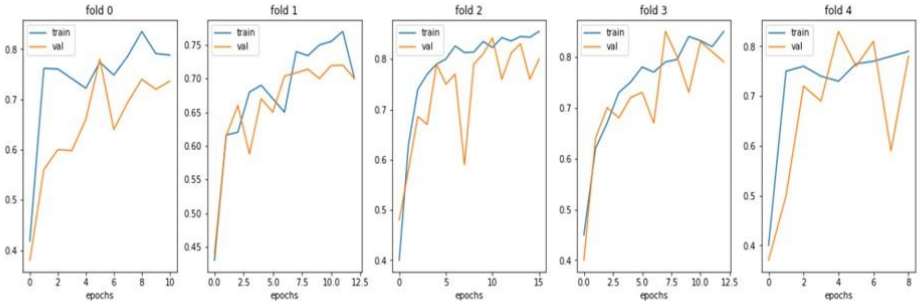
- У якості базових моделей обираємо EfficientNet-B4 та DenseNet169.
- Будуємо незважений ансамбль з 5 фолдів, які формуються з трансформованих та вихідних даних шляхом стратифікації.
- В якості критерія точності роботи використовуємо метрику Macro F1-score.
- Застосовуємо EarlyStopping та пониження швидкості навчання



Модель	fold 0	fold 1	fold 2	fold 3	fold 4	Average
EfficientNetB4	0.7866	0.8239	0.8627	0.7960	0.8121	0.8163
DenseNet169	0.7806	0.7294	0.8421	0.8542	0.8317	0.8076



EfficientNetB4



DenseNet169

Демонстрація роботи

Test samples

99 Iryna Oksiuta 0.813

image	complex	frog_eye_leaf_spot	powdery_mildew	rust	scab
85f8cb619c66b863.jpg	0.490224	0.724904	0.000142	0.001134	0.699495
ad8770db05586b59.jpg	0.182869	0.915205	0.000081	0.001823	0.212230
c7b03e718489f3ca.jpg	0.316361	0.049370	0.000496	0.020903	0.588541

	image	labels
0	85f8cb619c66b863.jpg	complex frog_eye_leaf_spot scab
1	ad8770db05586b59.jpg	frog_eye_leaf_spot
2	c7b03e718489f3ca.jpg	scab

Точність на тестовому наборі: **81, 3%**

Висновки:

- Розглянуто існуючі підходи до вирішення задач комп'ютерного зору.
- Побудовано модель розпізнавання категорій хвороб рослин.
- Вирішено питання розмаїття візуальних шумів, шляхом доповнення вихідного набору даних.
- Проведено експеримент для нових тестових зображень.

Подальші дослідження

- Покращення якості побудованої моделі, шляхом розширення вихідного набору даних, додавання більшої кількості варіацій зображень.
- Розширення простору можливих міток патологій яблунь або застосування отриманих підходів до інший плодових культур.
- Впровадження побудованої моделі в готові продукти. Робота в дуеті з засобами зйомки, наприклад, дронами.

Дякую за увагу!