НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

Завідувач кафедри

_____Олена ЧУМАЧЕНКО

«___» _____ 2023 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи і методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки»

на тему: «Застосування глибинного навчання для розпізнавання та класифікації лейкоцитів на зображеннях крові»

Виконав :
студент IV курсу, групи КІ-93
Крушельницький Ілля Михайлович _____

Керівник:
професор, д.т.н., Данилов Валерій Якович _____

Консультант з нормконтролю:
Гончарук Максим Миколайович _____

Консультант з економічного розділу:
доцент, к.е.н., Рощина Надія Василівна _____

Рецензент:
професор, д.т.н., Новіков Олексій Миколайович _____

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

Київ, 2023

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітня програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ:

Завідувач кафедри

_____Олена ЧУМАЧЕНКО

«___» _____ 2023 р.

ЗАВДАННЯ

на дипломну роботу студенту

Крушельницький Ілля Михайлович

1. Тема роботи «Застосування глибинного навчання для розпізнавання та класифікації лейкоцитів на зображеннях крові», керівник роботи професор, д.т.н. Данилов Валерій Якович, затверджені наказом по університету від «30» травня 2023 р. № 2065-с
2. Термін подання студентом роботи 08.06.2023
3. Вихідні дані до роботи – набір зображень лейкоцитів у крові під мікроскопом.
4. Зміст роботи – 1. Дослідження предметної області; 2. – Математичні основи робота; 3. Аналіз результатів роботи; 4. Функціонально-вартісний аналіз програмного продукту.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

6. Консультанти розділів роботи

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

7. Дата видачі завдання 21.02.2023

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Підготовка даних до роботи	27.02.2023	Виконано
2.	Вивчення літератури за темою роботи	25.03.2023	Виконано
3.	Підготовка першого розділу	10.04.2023	Виконано
4.	Розробка нейронних мереж	16.04.2023	Виконано
5.	Підготовка другого розділу	08.05.2023	Виконано
6.	Розробка веб-клієнта	20.05.2023	Виконано
7.	Підготовка третього розділу	24.05.2023	Виконано
8.	Підготовка економічної частини	01.06.2023	Виконано
9.	Підготовка презентації доповіді	03.06.2023	Виконано
10.	Оформлення дипломної роботи	05.06.2023	Виконано

Студент Ілля КРУШЕЛЬНИЦЬКИЙ

Керівник Валерій ДАНИЛОВ

АНОТАЦІЯ

Дипломна робота: 93 сторінки, 32 рисунки, 11 таблиць, 1 додаток, 24 джерела.

НЕЙРОННІ МЕРЕЖІ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, КЛАСИФІКАЦІЯ, ВЕБ-ДОДАТОК, ЛЕЙКОЦИТИ

Об’єкт дослідження – класифікація лейкоцитів на зображеннях крові під мікроскопом.

Мабуть кожна людина хоча б раз в житті здавала кров на аналізи. Для звичайної людини весь процес після здачі крові – всього лиш очікування результатів. Але «поза завісою» відбувається довгий процес, який потребує задіяння великої кількості спеціалістів: починаючи лаборантом, який зробить зображення крові під мікроскопом, закінчуючи кваліфікованим спеціалістом, який визначить тип лейкоцитів.

Мета роботи – створити якісне та просте рішення, засноване на веб-додатку, що дозволятиме дуже просто розпізнавати та класифікувати лейкоцити.

Програма базується на автономності, де додаток самостійно визначає категорії лейкоцитів, без необхідності втручання людей як носіїв знань. Це призведе до значного прискорення та покращення роботи медичної структури в цілому.

ABSTRACT

Bachelor thesis: 93 pages, 32 figures, 11 tables, 1 appendix, 24 sources.

NEURAL NETWORKS, CLASSIFICATION, CONVOLUTIONAL NEURAL NETWORKS, WEB APPLICATION, LEUKOCYTES.

Research Object – Classification of leukocytes in blood images under a microscope.

Perhaps every person has undergone blood tests at least once in their life. For an ordinary person, the entire process after giving blood is just waiting for the results. But behind the scenes, there is a process that involves the involvement of a large number of specialists, starting from a laboratory technician who captures blood images under a microscope, to a qualified specialist who identifies the type of leukocytes.

The goal of the study is to create a simple and flexible solution based on a web application that would allow for easy recognition and classification of leukocytes. The foundation of the program is the autonomy of the application from individuals who possess knowledge.

In other words, the application will automatically determine the categories of leukocytes. This will significantly speed up and improve the overall work of the medical institution.

ЗМІСТ

ВСТУП

Розпізнавання та класифікація лейкоцитів на зображеннях крові відіграє важливу роль у діагностиці різних захворювань. Білі кров’яні клітини, або лейкоцити, є основними елементами імунної системи. У залежності від типу та кількості лейкоцитів можна визначити наявність тих чи інших патологій в організмі людини.

На сьогоднішній день аналіз крові на наявність та визначення типу лейкоцитів виконується вручну. Звичайно це має свої недоліки у вигляді великої часозатратності та високої залежності від досвіду спеціаліста, який цим займається.

Мета цієї роботи полягає в застосуванні глибинного навчання для розпізнавання та класифікації лейкоцитів на зображеннях крові з використанням нейронних мереж.

Глибинне навчання є потужним інструментом машинного навчання, що дозволяє автоматично вивчати складні властивості зображень та робити точні класифікації.

Дослідження в цій області може мати значний практичний вплив на медичну діагностику та розпізнавання захворювань шляхом автоматичного аналізу зображень крові. Впровадження глибинного навчання для класифікації лейкоцитів може прискорити процес аналізу, покращити точність діагностики та сприяти ранньому виявленню захворювань.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність роботи

Точність та швидкість діагностики є критично важливими факторами для успішного лікування різних захворювань. Застосування глибинного навчання для розпізнавання та класифікації лейкоцитів на зображеннях крові відкриває нові перспективи для автоматизованого, об’єктивного та точного аналізу.

Розвиток штучного інтелекту та глибинного навчання відкриває шлях до персоналізованої медицини, де діагностика та лікування можуть бути налаштовані на індивідуальні особливості кожного пацієнта. Застосування глибинного навчання дозволяє створити моделі, які враховують унікальні характеристики кожної людини та забезпечують більш точну діагностику та прогнозування.

Розвиток цієї технології сприяє прискоренню процесу аналізу крові та зниженню навантаження на медичний персонал. Замість ручної класифікації, що вимагає багато часу та концентрації, системи глибинного навчання можуть автоматизувати цей процес, забезпечуючи швидкі та точні результати, що прискорює прийняття рішень та полегшує роботу медичного персоналу.

Враховуючи ці фактори, актуальність роботи очевидна. Вона обіцяє змінити парадигму медичної діагностики, роблячи її більш точною, швидкою, персоналізованою та ефективною.

Такі технології мають потенціал зберегти багато життів, покращити якість діагностичної допомоги та сприяти розвитку медичної науки та практики в цілому.

Усі вищезгадані факти призводять до термінової необхідності розробки більш адаптивних та практичних систем високопродуктивного розпізнавання лейкоцитів.

1.2 Опис предметної області

1.2.1 Білі кров’яні клітини

Біла кров’яна клітина, також відома як лейкоцит або біла корпускула, є клітинним компонентом крові, що містить ядро, може рухатися та не має гемоглобіну. Основним завданням лейкоцитів є захист організму від інфекцій та хвороб. Білі кров’яні клітини можуть виробляти антитіла, знищувати інфекційні агенти та ракові клітини, а також поглинати чужородні матеріали та клітинні залишки для виконання захисних функцій. Лейкоцити, як правило, знаходяться у циркуляції, але їх також можна зустріти в тканинах, де вони борються з інфекціями [1].

Існує кілька типів лейкоцитів, кожен з яких відповідає за різні аспекти.

1.2.2 Лімфоцити

Лімфоцити допомагають організму боротися з інфекціями. Вони присутні в імунній системі всіх хребетних. У дорослих людей лімфоцити становлять приблизно 20-40 відсотків загальної кількості білих кров’яних клітин. Лімфоцити можна знайти [2]:

- У венах і артеріях (у кровообігу організму)
- У вузлах та каналах лімфатичної системи організму.
- Розсіяні по всьому тілу, наприклад, в селезінці, мигдалинах, кишечнику та на слизовій оболонці дихальних шляхів. Тут лімфоцити представляють так звану «лімфоїдну тканину».
- У кишечнику. Тут лімфоцити організовані більш структуровано, утворюючи фолікули (вони є важливою частиною імунної системи, контролюючи бактерії, що проживають у кишечнику, та запобігаючи їх розмноженню).

Існують три основних типи лімфоцитів: В-клітини, Т-клітини та клітини натуральних вбивць [2].

Т-лімфоцити відрізняються від В-лімфоцитів і клітин натуральних вбивць тим, що на їхній клітинній мембрані є білок, який називається Т-клітинним рецептором. Т-клітинні рецептори здатні впізнавати різні типи специфічних антигенів (речовин, що викликають імунну відповідь). На відміну від В-лімфоцитів, Т-клітини не використовують антитіла для боротьби з мікробами.

Т-клітини виникають у кістковому мозку і дозрівають у тимусі. У тимусі Т-клітини розмножуються і диференціюються на допоміжні, регуляторні або цитотоксичні, або стають пам'яттю Т-клітин. Потім вони направляються до периферичних тканин або циркулюють у крові, або лімфатичній системі. Під впливом відповідного антигену, допоміжні Т-клітини виділяють цитокіни, які стимулюють диференціацію В-клітин у плазматичні клітини (клітини, що виробляють антитіла). Регуляторні Т-клітини, як вказує їхня назва, контролюють імунні реакції. Цитотоксичні Т-клітини, які активуються різними цитокінами, зв'язуються та усувають інфіковані клітини і клітини раку.

В-клітини з'єднуються з антигенами безпосередньо на поверхні вірусу або бактерії, тоді як Т-клітини з'єднуються з антигенами вірусу на зовнішній стороні інфікованих клітин. Наш організм має до 10 мільярдів різних В-клітин [2].

Коли рецептор В-клітини з'єднується зі своїм конкретним антигеном, допоміжна Т-клітина випускає речовини, які призводять до того, що В-клітина ділиться багато разів. Це створює армію В-клітин з ідеально сформованим рецептором В-клітини, який з'єднується зі зловмисником, тобто антигеном. Багато з цих В-клітин швидко перетворюються на плазматичні клітини.

Плазматичні клітини виробляють і вивільняють антитіла, які з'єднуються з тим самим антигеном, що й оригінальний рецептор В-клітини. Плазматичні клітини виробляють тисячі антитіл за секунду, які розповсюджуються по всьому організму, затримуючи будь-які віруси, які вони зустрічають по дорозі.

Неактивовані В-клітини циркулюють у крові до того моменту, коли вони зіткнуться з антигеном і активуються [2].

Після активації В-клітини виробляють антитіла, необхідні для боротьби з інфекцією.

В-клітини необхідні для адаптивного або специфічного імунітету, який спрямований на знищення іноземних загарбників, які проникли через початкові захисні бар'єри організму. Адаптивні імунні відповіді є вельми специфічними та забезпечують довготривалий захист від патогенів, що викликають ці реакції.

Клітини-вбивці (NK-клітини) є частиною імунної системи, яку людина має з народження. Вони захищають організм від пухлин і від клітин, заражених вірусами. NK-клітини активуються за допомогою речовин, які виділяються пошкодженими клітинами і називаються інтерферонами.

NK-клітини вміють відрізняти інфіковані клітини, пухлини і нормальні клітини. Вони розпізнають зміни в рівні молекули на поверхні, яка називається ВГК (великий-гістосумісний комплекс).

Потім NK-клітини вивільняють цитотоксичні (клітини-вбивці) гранули, які знищують змінені клітини [2].

1.2.3 Нейтрофіли

Нейтрофіли – це тип білих кров'яних клітин, які характеризуються можливістю бути забарвленими нейтральними фарбниками та боротьбою із інфекційними мікроорганізмами.

Нейтрофіли, разом з еозинофілами та базофілами, утворюють групу білих кров'яних клітин, відомих як гранулоцити. Гранули нейтрофілів зазвичай забарвлюються в рожевий або фіолетово-синій колір після обробки фарбником. Приблизно 50-80 відсотків усіх білих кров'яних клітин в організмі людини є нейтрофілами [3].

Ядро складається з двох до п'яти долонь, з'єднаних волоскоподібними нитками. Нейтрофіли рухаються амебоподібним чином. Вони витягують довгі виступи, які називаються псевдоподіями, у які потрапляють їх гранули. Потім слідує скорочення ниток, що базуються в цитоплазмі, що притягує ядро та задню частину клітини вперед. Таким чином, нейтрофіли швидко рухаються по поверхні.

Кістковий мозок звичайної дорослої людини щоденно виробляє приблизно 100 мільярдів нейтрофілів. Формування зрілого нейтрофіла з клітин-попередників в кістковому мозку займає приблизно один тиждень; однак, після переходу в кров, зрілі клітини живуть всього кілька годин або трохи довше після міграції в тканини. Щоб запобігти швидкому виснаженню короткоживучих нейтрофілів (наприклад, під час інфекції), кістковий мозок утримує велику кількість у запасі для застосування у відповідь на запалення або інфекцію.

У межах організму нейтрофіли мігрують до зон інфекції або ушкодження тканин. Сила притягання, що визначає напрямок руху нейтрофілів, відома як хемотаксис і пов’язується з речовинами, що вивільняються на місцях ушкодження тканин [3].

З багатьох нейтрофілів, що циркулюють поза кістковим мозком, половина знаходиться в тканинах, а половина – у кровоносних судинах; з тих, що знаходяться в кровоносних судинах, половина перебуває в основному кровотоці, а інша половина повільно рухається по внутрішніх стінках кровоносних судин, готова увійти в тканини після отримання хемотактичного сигналу від них.

Нейтрофіли активно здійснюють фагоцитоз: вони поглинають бактерії та інші мікроорганізми та мікроскопічні частинки. Гранули нейтрофілів – це мікроскопічні пакети потужних ферментів, здатних розщеплювати багато видів клітинних матеріалів. Коли бактерія поглинається нейтрофілом, вона оточується фагоцитарною вакуолею, обсадженою зворотним вклиненням мембрани, структурою, відомою як фагосом. Гранули випускають свій вміст у вакуолю, що містить організм. Під час цього гранули нейтрофіла вичерпуються (дегрануляція). Метаболічний процес в гранулах виробляє перекис водню та високоактивну форму кисню (супероксид), які знищують поглинуті бактерії. Завершення знишення шкідливого організму здійснюється ферментами [3].

1.2.4 Еозинофіли

Еозинофіли – це тип лейкоцитів, що характеризуються здатністю гістологічно барвитися кислими фарбами (наприклад, еозином). Разом із базофілами та нейтрофілами, еозинофіли належать до групи білих кров’яних клітин, відомих як гранулоцити.

Еозинофіли містять великі гранули, а ядро складається з двох нерозділених долоньок. Крім того, гранули еозинофілів зазвичай фарбуються червоною фарбою, внаслідок чого їх легко відрізнити від інших гранулоцитів при огляді зразків крові під мікроскопом.

Еозинофіли є рідкісними, становлячи менше 1 відсотка загальної кількості білих кров’яних клітин у людському організмі. Еозинофіли, подібно до інших гранулоцитів, утворюються в кістковому мозку, поки не попадуть до кровообігу. Еозинофіли покидають кровообіг упродовж годин після вивільнення з кісткового мозку і мігрують у тканини (зазвичай шкіри, легенів і дихальних шляхів) через лімфатичні канали.

Подібно до нейтрофілів, еозинофіли реагують на хемотактичні сигнали, що вивільняються на місці пошкодження клітин. Ці хімічні сигнали орієнтують еозинофіли та стимулюють їх міграцію в напрямку пошкоджених клітин. Еозинофіли активно рухаються та фагоцитують, беруть участь у гіперчутливих і запальних реакціях, головним чином, заглушуючи їх деструктивний вплив.

Еозинофіли також беруть участь у захисті від паразитів. Еозинофіли та антитіла класу імуноглобуліну Е працюють разом для знищення паразитів, таких як плоскі черви, що викликають шистосомоз. Еозинофіли прилипають до червів, і вивільняють хімічні речовини зі своїх гранул, які розкладають жорстку захисну шкіру паразита.

1.2.5 Моноцити

Моноцити є важливою складовою вродженої імунної системи. Це тип білих кров'яних клітин, який диференціюється в макрофаги та дендритні клітини

для регулювання клітинного гомеостазу, особливо в умовах інфекції та запалення [5].

Моноцити виконують дві важливі ролі: регулярно патрулюють організм у пошуках мікробних клітин та організовують імунну відповідь у разі інфекції та запалення. На поверхні моноцитів знаходяться толоподібні рецептори, які взаємодіють з PAMP (молекулярними паттернами, специфічними для патогенів), що знаходяться на мікробних клітинах. У відповідь на такі подразники, моноцити мігрують з кісткового мозку до кровообігу і проникають в тканини протягом 12-24 годин.

Для проникнення в уражені ділянки моноцити спочатку фіксуються на ендотелії, потім повільно рухаються по поверхні судин. У подальшому моноцити тісно прикріплюються до ендотелію і, нарешті, проходять через ендотеліальні клітини за допомогою процесу, який називається діапедез. Після цього моноцити можуть проникнути через базальну мембрану ендотелію і мігрувати до зони запалення. Цей процес відбувається за участю молекул як на поверхні моноцитів, так і на поверхні ендотеліальних клітин [5].

Залежно від функції, різні моноцити можуть виділяти різні типи зростаючих факторів та цитокінів. Моноцити функціонують як фагоцити та клітини, що представляють антигени в периферичній крові, засмоктуючи і усуваючи мікроорганізми, чужорідні матеріали та мертві або пошкоджені клітини. Клітини, які представляють антигени, включають макрофаги, дендритні клітини, В-лімфоцити та активовані ендотеліальні клітини.

Макрофаги можуть подальше диференціюватися у спеціалізовані макрофаги залежно від їх розташування. Серед них гістіоцити в сполучній тканині, мікроглія в мозку, остеокласти в кістці, мезангіальні клітини в нирках та альвеолярні макрофаги в легенях. Моноцити можуть виділяти цитокіни, які залучають додаткові клітини та білки в уражену ділянку для масштабної імунної відповіді. Типи вивільнюваних цитокінів будуть варіюватися залежно від функції. Для відновлення гомеостазу моноцити можуть сприяти перебудові та загоєнню за допомогою протизапальних цитокінів [5].

1.3 Опис та аналіз даних

Для тренування моделі використовувався набір даних, доступний в загальному доступі на безкоштовній платформі Kaggle [6].

Датасет містить 12 500 зображень кров'яних клітин, що представлені у вигляді зображень формату JPEG. Кожне зображення має розмір 640x480 пікселів, а також мітку класу, що вказує на тип клітини, яку воно зображує (рисунок 1.1). Мітки типів клітин зберігаються у файлі формату CSV.

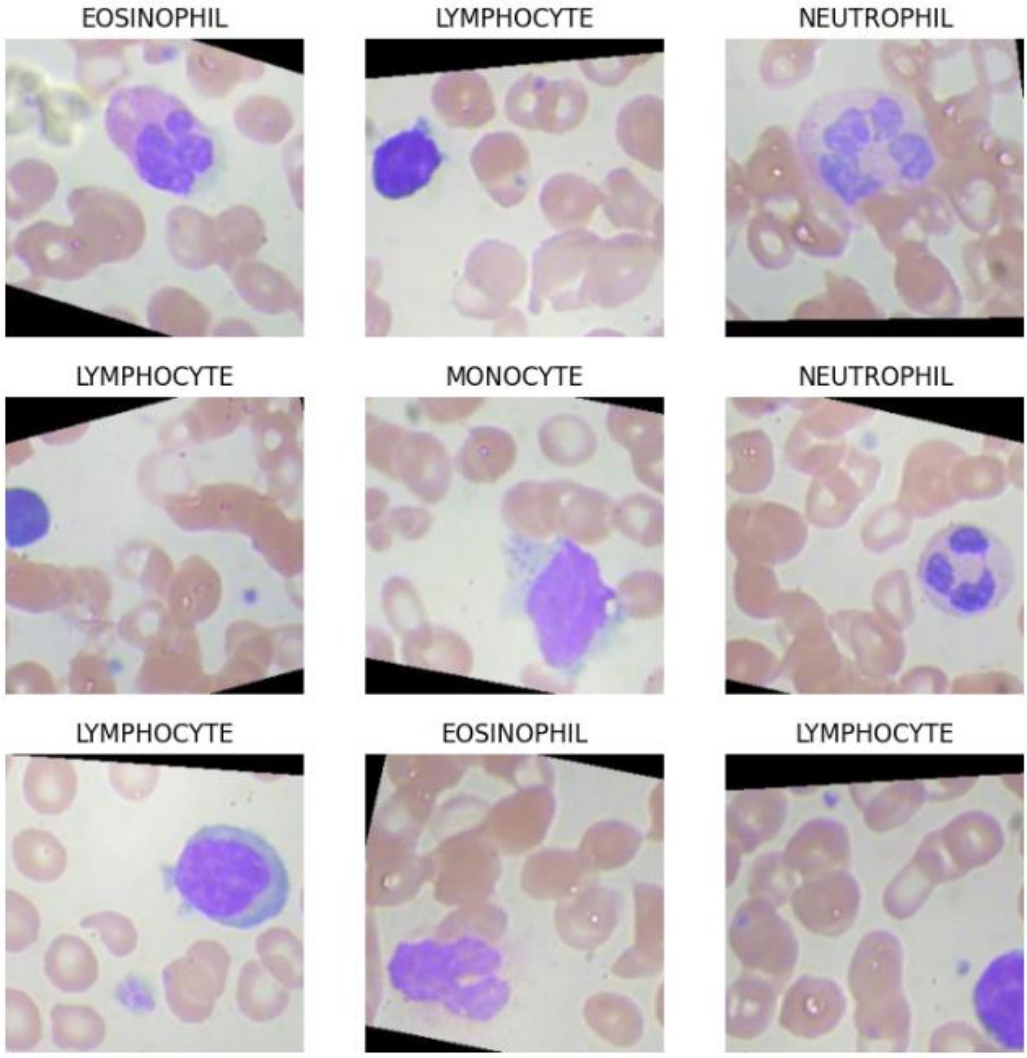

Рисунок 1.1 – Зображення лейкоцитів із мітками класів

Кількість зображень для кожного класу приблизно однакова, оскільки є близько 3000 зображень для кожного класу. Початково зображення не були

поділені на тренувальні, валідаційні та тестувальні. Розділення здійснюється після завантаження даних до середовища програмування.

Кількість зображень для тренування, валідації та тестування розподіляється рівномірно для кожного із класів.

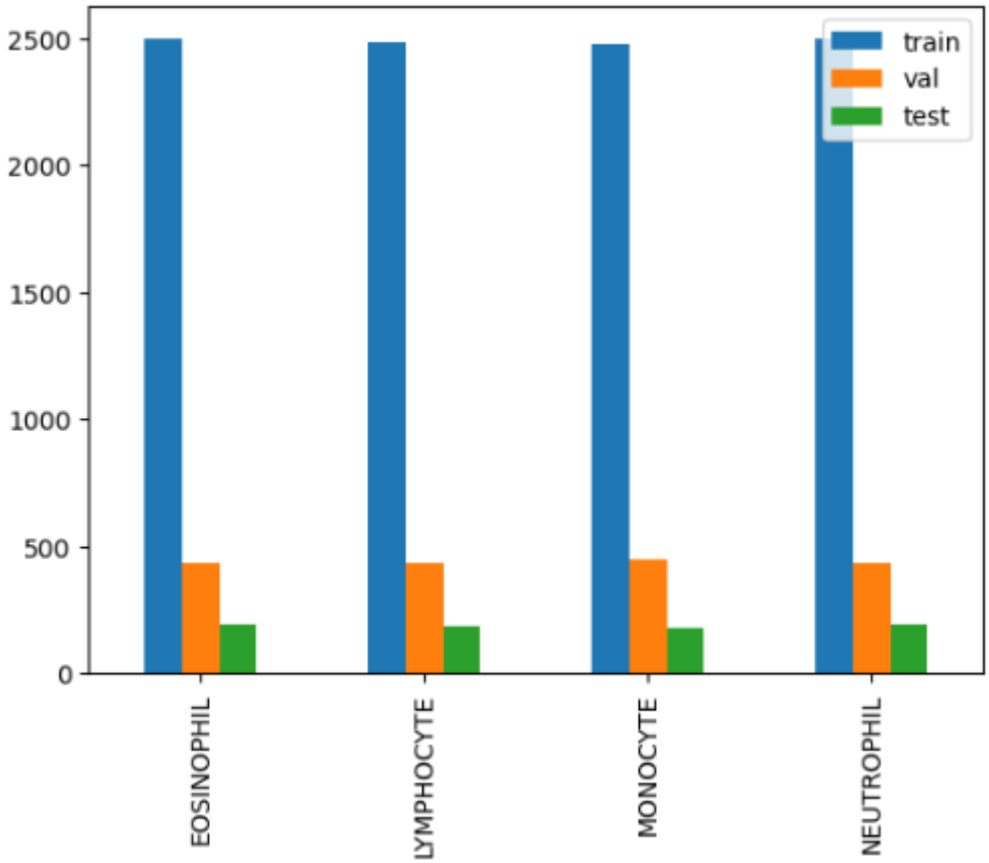

Рисунок 1.2 – Розподіл зображень за категоріями

Як видно із рисунку 1.2 дисбалансу у розподілі немає. Точну кількість зображень відображає таблиця 1.1.

Таблиця 1.1

Точна кількість зображень для кожного класу

	training	validation	test
EOSINOPHIL	2497	433	190
LYMPHOCYTE	2483	432	188
MONOCYTE	2478	444	176
NEUTROPHIL	2499	432	192

1.4 Постановка задачі

На початку потрібно визначитися з типом задачі, яку ми вирішуємо. Наша мета – класифікація кожного зображення на один з чотирьох класів: Eosinophil, Lymphocyte, Monocyte, Neutrophil.

Таким чином, ми стикаємося з задачею класифікації зображень.

Задача класифікації зображень полягає в тому, щоб розподілити множину об’єктів (фотографій, малюнків і т. д.) на класи згідно з певною логікою. У нас є скінченна множина об’єктів, про які відомо, до яких класів вони належать.

Важливим елементом є дослідження вже існуючих архітектур для вирішення даної задачі.

Необхідно розробити модель машинного навчання для автоматичної класифікації зображень об’єктів на класи. Мета полягає в тому, щоб модель змогла правильно визначати клас кожного зображення з вибірки і вказувати тип об’єкта на зображенні.

Задача класифікації зображень буде вирішуватися за допомогою методів машинного навчання, зокрема штучних згорткових нейронних мереж у вигляді навчання з учителем. Для досягнення цієї мети буде використана відповідна навчальна вибірка, валідаційна вибірка та тестова вибірка для оцінки якості моделі[7].

Тренувальна вибірка використовується для налаштування (оптимізації) ваг і коефіцієнтів моделі.

Валідаційна вибірка використовується для підбору параметрів, ознак та прийняття рішень, пов’язаних з навчанням моделі, наприклад, уникнення перенавчання.

Тестова вибірка використовується для оцінки якості навчання моделі і не повинна залежати від тренувальної.

Також необхідно розробити цілу систему, яка буде максимально зручною для користувача. Основні вимоги до системи полягають у тому, щоб вона працювала на сервері, не вимагала від користувача завантаження додаткових модулів або бібліотек та мала зрозумілий програмний інтерфейс для майбутньої інтеграції з іншими системами, що вимагають розпізнавання лейкоцитів.

Результатом роботи системи буде виведення на екран назви категорії, до якої належить об’єкт на зображенні.

1.5 Висновки до розділу 1

У даному розділі розглянуто актуальність задачі розпізнавання та класифікації лейкоцитів.

Зроблено детальний огляд на предметну область, внаслідок чого вдалось визначити за що відповідає кожен із типів лейкоцитів.

Проведено опис та аналіз даних, з поясненням їх походженням, кількості, розмірів та типів. Наведені рисунки та таблиця для кращої візуалізації інформації.

У кінці розділу розглянуто поняття класифікації зображення, валідаційної, тестової та тренувальної вибірок, а також поставлено задачу та базові вимоги до системи, що розроблюється.

РОЗДІЛ 2 МАТЕМАТИЧНІ ОСНОВИ РОБОТИ

2.1 Згорткові нейронні мережі

Згорткова нейронна мережа, також відома як CNN або ConvNet, є класом нейронних мереж, спеціалізованих у обробці даних з сітково-подібною топологією, наприклад, зображень. Цифрове зображення є бінарним представленням візуальних даних. Воно містить серію пікселів, розташованих у вигляді сітки, які мають значення, що вказують на яскравість та кольори кожного пікселя (рис. 2.1).

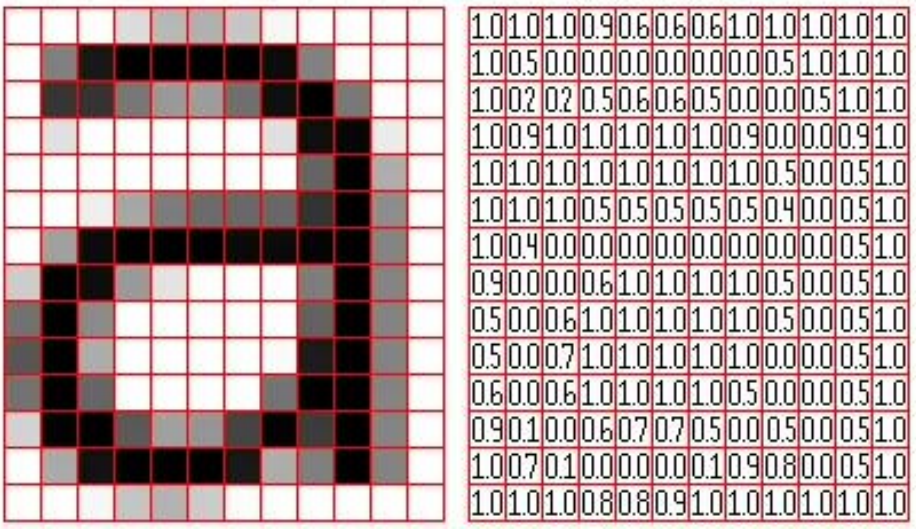

Рисунок 2.1 – Представлення зображення у вигляді сітки пікселів [8]

Людський мозок обробляє величезний обсяг інформації в момент, коли ми бачимо зображення. Кожен нейрон працює в своєму рецептивному полі і пов’язаний з іншими нейронами таким чином, що вони охоплюють усе зорове поле. Як і кожен нейрон в людському мозку реагує на стимули лише в обмеженій області зорового поля, так і кожен нейрон у згортковій нейронній мережі також обробляє дані лише у своєму рецептивному полі. Шари розташовані таким чином, щоб вони спочатку виявляли простіші патерни (лінії, криві і т. д.), а потім більш складні патерни (обличчя, об’єкти і т. д.). Використовуючи згорткову нейронну мережу, можна надати комп’ютерам здатність бачити [8].

2.2 Архітектура згорткових нейронних мереж

Архітектура згорткової нейронної мережі (CNN) загалом складається з декількох типів шарів (рис. 2.2): вхідний шар (input layer), згорткові шари (convolutional layers), шари активації (activation layer), пулінгові шари (pooling layers) та повнозв’язані шари (fully connected layers).

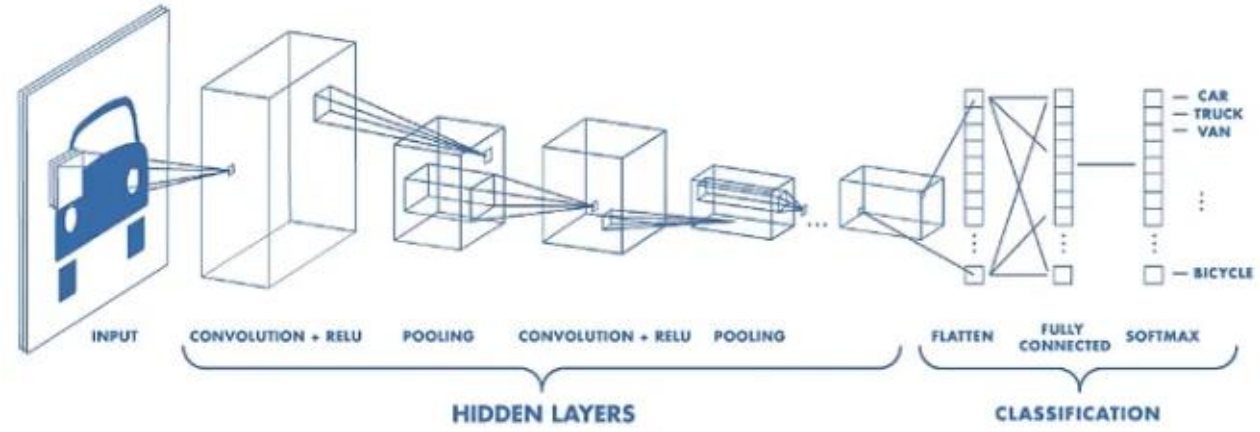

Рисунок 2.2 – Архітектура згорткової нейронної мережі [8]

2.2.1 Input layer

У Convoluted Neural Network (CNN) вхідний шар – це перший шар, який отримує вхідне зображення. На відміну від стандартної нейронної мережі, вхідний шар CNN розроблений з урахуванням просторової структури зображення і розташований у 3D-об'ємі у трьох вимірах: ширині, висоті та глибині (глибина відноситься до третього виміру об'єму, такого як кількість каналів у зображенні або кількість фільтрів у шарі) [9].

2.2.2 Convolution layer

У згортковій нейронній мережі (CNN) згортковий шар є основним блоком мережі і місцем проведення більшості обчислень. Цей шар виконує скалярний

добуток між двома матрицями, де одна матриця є набором параметрів, які вивчаються, відомих як ядро, а інша матриця є обмеженою частиною приймального поля. Ядро просторово менше за зображення, але має більшу глибину. Це означає, що якщо зображення складається з трьох (RGB) каналів, висота і ширина ядра будуть просторово маленькими, але глибина охоплює всі три канали.

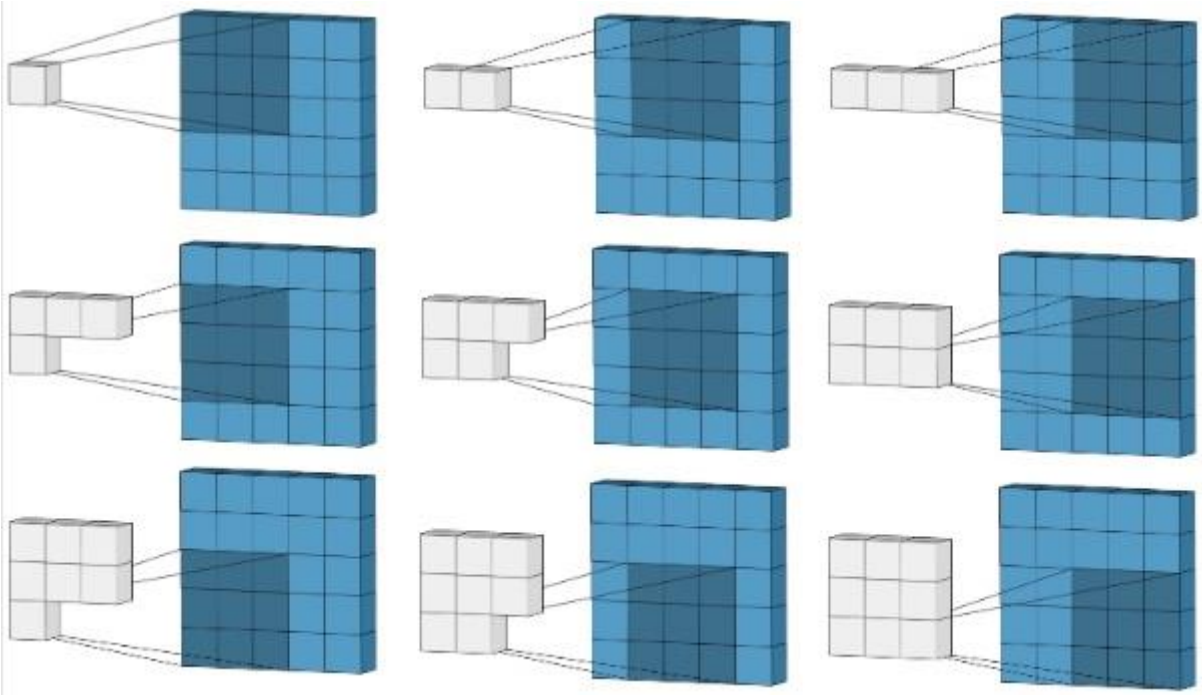

Рисунок 2.3 – Ілюстрація конволюційної операції [8]

Під час прямого проходження ядро ковзає вздовж висоти та ширини зображення, створюючи зображення, що відображає відповідний регіон приймання (рис. 2.3). Це створює двовимірне представлення зображення, відоме як карта активації, яка показує відповідь ядра в кожній просторовій позиції зображення. Розмір ковзання ядра називається кроком (stride) та може змінюватись в залежності від завдання [8].

Якщо у нас є вхід розміром H x W x D (висота, ширина, глибина відповідно) і кількість ядер з просторовим розміром F, кроком S та значенням заповнення P, то розмір вихідного обсягу можна визначити за наступною формулою (2.1):

$$W_{out} = \frac{W - F + 2P}{S} + 1 \qquad (2.1)$$

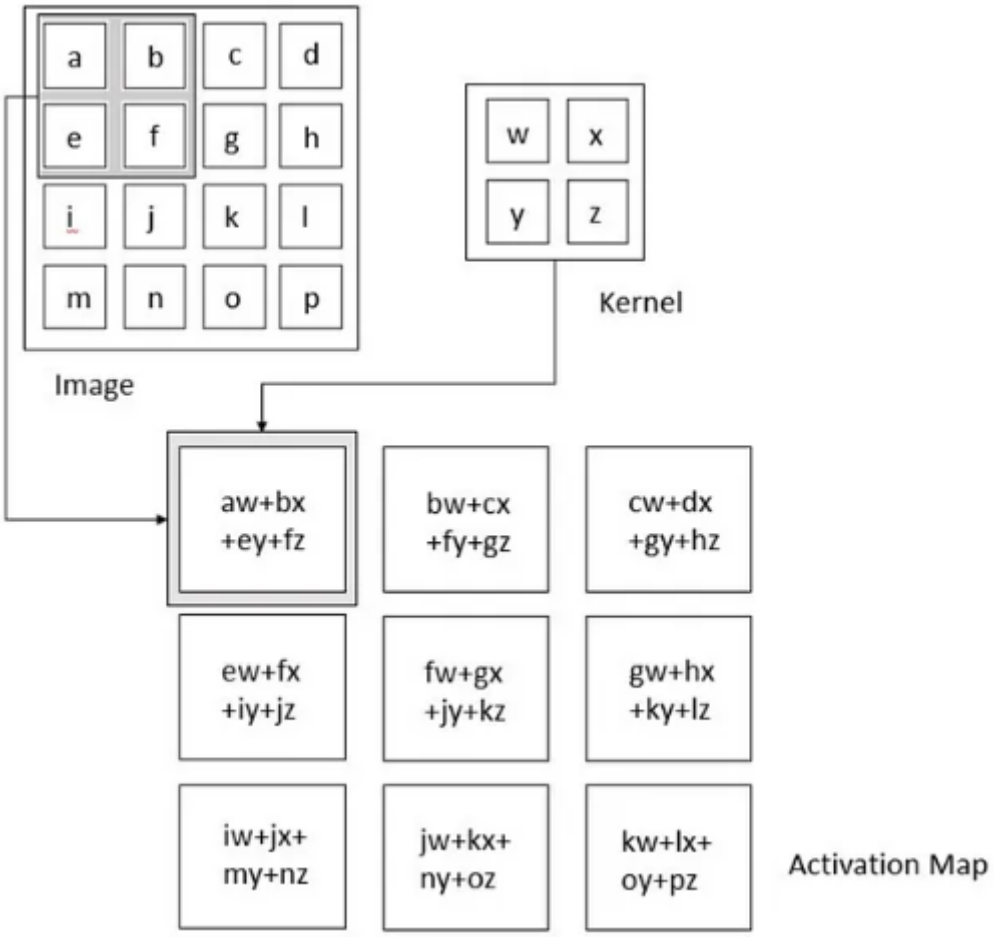

Рисунок 2.4 – Карта ознак після конволюційної операції [8]

Результатом виконання конволюційної операції буде карта ознак (рис. 2.4).

2.2.3 Activation layer

Оскільки згортка є лінійною операцією, а зображення далекі від лінійності, після згорткового шару часто розташовують нелінійний шар, щоб внести нелінійність до карти активації. Існує кілька типів нелінійних операцій, популярні з них такі [8]:

- Sigmoid
- Tanh
- ReLU
- Softmax

2.2.3.1 Sigmoid

Нелінійність сигмоїду має наступну математичну формулу (2.2):

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

Вона приймає дійсне число і «стискає» його до діапазону між 0 і 1. Однак, дуже небажаною властивістю сигмоїду є те, що коли активація знаходиться на одному з країв, градієнт стає майже нульовим. Якщо локальний градієнт стає дуже малим, то в процесі зворотнього поширення помилки він фактично «пригнічує» градієнт. Крім того, якщо дані, що надходять в нейрон, завжди позитивні, то вихід сигмоїду буде складатися з одних позитивних або всіх негативних значень (рис. 2.5), що призводить до зигзагоподібної динаміки оновлення градієнтів для ваг [8].

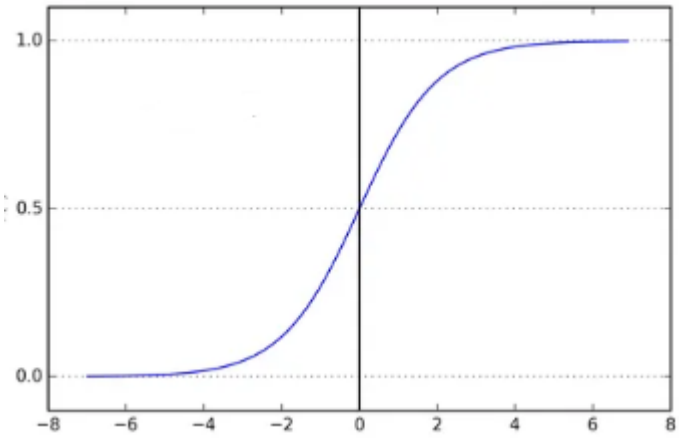

Рисунок 2.5 – Графік сигмоїдної функції [9]

2.2.3.2 Tanh

Tanh (гіперболічний тангенс) схожий на логістичну сигмоїду, але він працює краще. Діапазон функції tanh знаходиться від (-1 до 1). Tanh також має форму сигмоїди (форму букви «S»). Розраховується за формулою (2.3).

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.3)$$

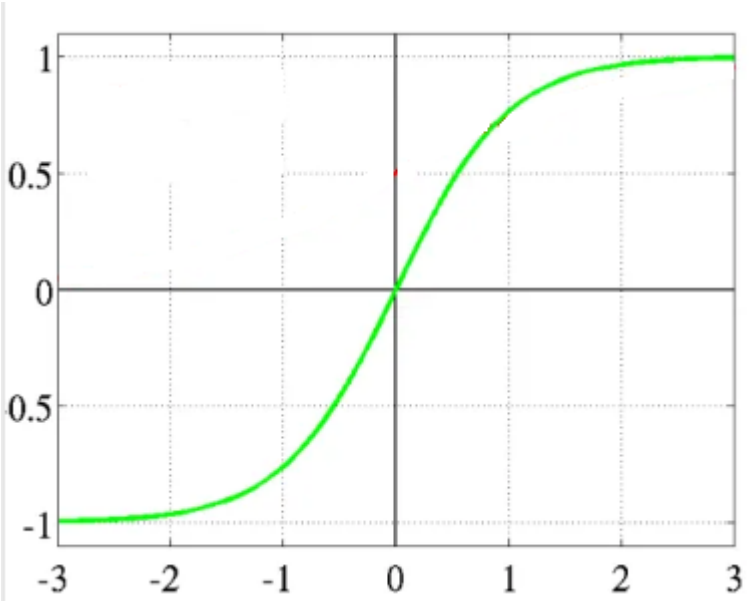

Рисунок 2.6 – Графік гіперболічного тангенсу [9]

Перевагою tanh є те, що від'ємні вхідні значення будуть відображені як дуже від'ємні, а нульові вхідні значення будуть відображені близько до нуля на графіку tanh (рис. 2.6). Функція tanh є диференційованою. Функція є монотонною, але похідна не є монотонною. Функція tanh в основному використовується для класифікації між двома класами. Вона може використовуватись як активаційна функція у шарах нейронних мереж, що використовуються для бінарної класифікації. Значення tanh може вказувати на належність до одного з класів, де значення близьке до 1 вказує на один клас, а значення близьке до -1 вказує на інший клас [9].

2.2.3.3 ReLU

ReLU (Rectified Linear Unit) є найбільш використовуваною функцією активації в даний час. Вона використовується майже в усіх згорткових нейронних мережах та глибокому навчанні. Обраховується за формулою (2.4)

$$f(x) = \max(0, x) \quad (2.4)$$

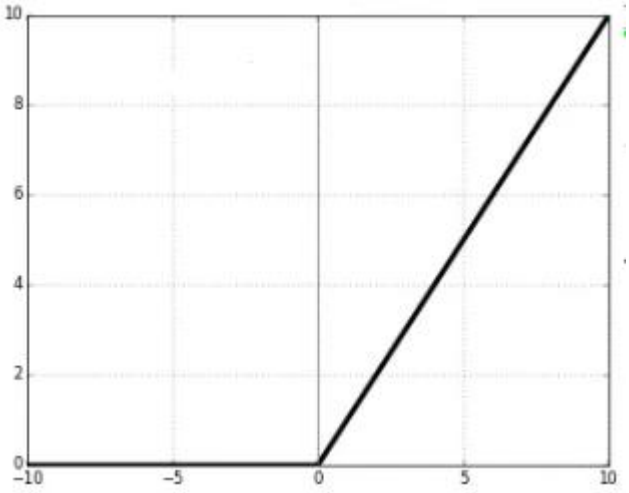

Рисунок 2.7 – Графік функції ReLU [9]

Як можна бачити із рисунку 2.7, функція ReLU є напіввипрямленою (знизу). Значення дорівнює нулю, коли x менше нуля, і $f(x)$ дорівнює x, коли x більше або дорівнює нулю. Діапазон становить від 0 до нескінченності. Функція та її похідна є монотонними.

На жаль, недолік полягає в тому, що під час навчання ReLU може бути вразливим. Великий градієнт, що протікає через нього, може оновити його таким чином, що нейрон більше не буде оновлюватися. Однак ми можемо працювати з цим, встановивши відповідну швидкість навчання [9].

2.2.3.4 Softmax

Softmax є типом функції активації, який зазвичай використовується в останньому шарі нейронної мережі для класифікаційних завдань.

Функція softmax приймає вектор вхідних значень і генерує вектор виходу тієї ж самої розмірності, де кожен елемент вектора виходу представляє ймовірність належності вхідного вектора до певного класу. Softmax зазвичай використовується разом з функцією втрати перехресної ентропії для тренування нейронної мережі.

2.2.4 Pooling layer

Одна з основних проблем згорткових шарів полягає в тому, що карта ознак, створена фільтром, залежить від місцерозташування. Це означає, що під час навчання згорткові нейронні мережі навчаються пов’язувати наявність певної ознаки з певним місцем на вхідному зображенні. Це може суттєво погіршити продуктивність. Натомість, ми хочемо, щоб карта ознак і мережа були незалежними від зсуву (що означає, що місце розташування ознаки не має значення).

У згортковій нейронній мережі зазвичай застосовується пулінг до карти ознак, створеної попереднім згортковим шаром, після чого використовується нелінійна функція активації [10].

Основна процедура пулінгу дуже схожа на операцію згортки. Фільтр ковзає по вихідній карті ознак попереднього згоркового шару. Найпоширеніший розмір фільтра 2×2, і його ковзання по вхідному зображенню здійснюється з кроком 2. Залежно від типу операції пулінгу, фільтр, що використовується, обчислює вихід на рецептивному полі (частині карти ознак, що знаходиться під фільтром). Існує кілька підходів до пулінгу. Найпоширеніші підходи – максимальний пулінг (max-pooling) і середнє значення пулінгу (average pooling).

У максимальному пулінгу фільтр просто вибирає найбільше значення пікселя в рецептивному полі. Наприклад, якщо у вас є 4 пікселя у полі зі значеннями 3, 9, 0 і 6, ви вибираєте 9 (рис. 2.8).

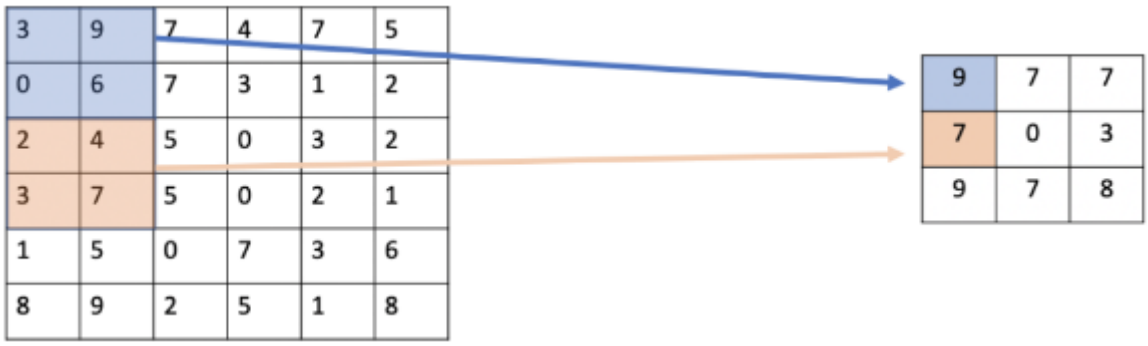

Рисунок 2.8 – Приклад роботи максимального пулінгу[10]

У середньому пулінгу розраховується середнє значення пікселів у рецептивному полі. Наприклад, за даними 4 пікселями із значеннями 3, 9, 0 та 6, шар середнього пулінгу буде мати вихідне значення 4,5. Заокруглення до цілих чисел дає нам 5 (рис. 2.9).

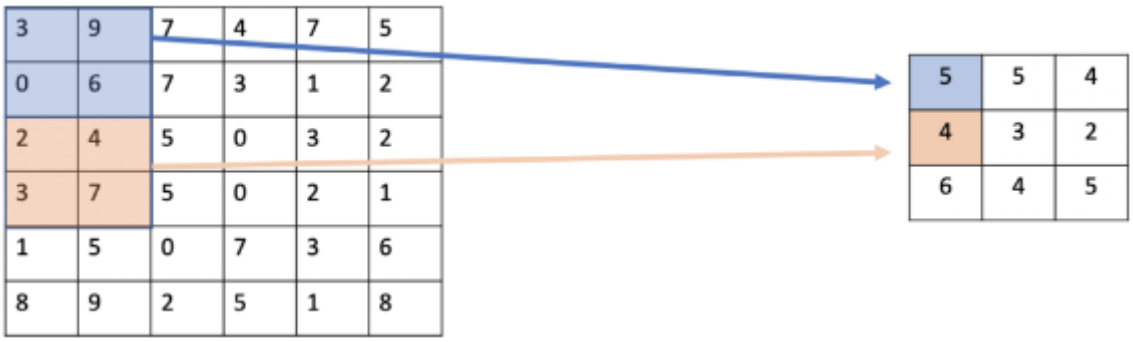

Рисунок 2.9 – Приклад роботи середнього пулінгу[10]

Можна розглядати числа, що обчислюються та зберігаються під час пулінгу, як показники наявності певної ознаки. Якби нейронна мережа опиралася лише на початкову карту ознак, її здатність виявляти ознаку залежала б від місцерозташування на карті.

Наприклад, якщо число 9 зустрічалося лише у верхньому лівому квадраті, мережа навчилася б пов’язувати ознаку саме з верхнім лівим квадратом. Застосування пулінгу дозволяє витягти цю ознаку у меншу, загальну карту, яка лише вказує на наявність або відсутність ознаки у певному квадраті. З кожним додатковим шаром карта зменшується, зберігаючи лише важливу інформацію про наявність цікавих ознак. При зменшенні розміру карти вона стає все більш незалежною від місцерозташування ознаки. Якщо ознака була виявлена в приблизній околиці початкового місця розташування, вона повинна подібно відображатися у карті, створеній пулінговими шарами[10].

Максимальний пулінг, завдяки своєму фокусу на екстремальних значеннях, звертає увагу на виразні ознаки та ребра в рецептивному полі. З іншого боку, середнє значення пулінгу створює більш плавну карту ознак, оскільки воно обчислює середні значення замість вибору екстремальних значень.

У практиці максимальний пулінг застосовується частіше, оскільки загалом краще впізнає видимі ознаки. У практичних застосуваннях середнє значення пулінгу використовується лише для зменшення розміру карт ознак.

Завдяки здатності згорткових шарів до зведення карт ознак, пулінг також допомагає класифікувати зображення різних розмірів. Класифікаційний шар нейронної мережі очікує вхідні дані у тому ж форматі. Звичайно, ми подаємо зображення одного стандартного розміру. Змінюючи зсуви під час операції пулінгу, ми можемо узагальнити зображення різних розмірів і все ж отримати карту ознак приблизно одного розміру.

Загалом, пулінг особливо корисний, коли у є завдання класифікації зображень, де потрібно просто виявити наявність певного об’єкта на зображенні, але не цікаво, де саме він розташований. Той факт, що фільтри пулінгу використовують більший крок, ніж згорткові фільтри, і призводять до менших виходів, також підтримує ефективність мережі і прискорює навчання. Іншими словами, незалежність від місцерозташування може значно покращити статистичну ефективність мережі [10].

2.2.5 Flatten layer

Після згорткових та пулінгових шарів наступним кроком є перетворення вихідного значення останнього пулінгового шару у одновимірний вектор за допомогою шару Flatten. Flatten-шар використовується для перетворення вихідного значення згорткових та пулінгових шарів з формату тензору 2D або 3D у одновимірний вектор, який можна подати на повністю зв’язаний шар (fully connected layer). Одновимірний вектор формується шляхом конкатенації всіх карт ознак у останньому пулінговому шарі. Flatten-шар не має навчальних параметрів, його єдиним завданням є зміна форми вхідного тензора[11].

2.2.6 Fully connected layer

Fully connected layer є останнім шаром мережі, який виконує остаточне завдання класифікації або регресії. Повністю зв’язаний шар складається з нейронів, які зв’язані з усіма активаціями в попередньому шарі [12].

Вхідними даними для повністю зв’язаного шару є вихідний результат останнього пулінгового або згорткового шару, який після цього згладжується і передається на вхід повністю зв’язаному шару. Отриманий плаский вихід потім з’єднується з одним або кількома fully connected layers, які виконують ті ж математичні операції, що й у штучних нейронних мережах. У кожному шарі повністю зв'язаної мережі вхідний вектор множиться на матрицю ваг і додається вектор зсуву, а результат проходить через функцію активації ReLU. Вихід останнього повністю зв'язаного шару проходить через функцію активації softmax.

Основна мета повністю зв'язаного шару полягає в тому, щоб вивчити нелінійні комбінації ознак, які видобуваються зі згорткових та пулінгових шарів. По суті, згорткові та пулінгові шари надають значущий, низькорозмірний і в певній мірі інваріантний простір ознак, а повністю зв'язаний шар вивчає (можливо, нелінійну) функцію у цьому просторі. Повністю зв'язаний шар дозволяє здійснювати навчання всього процесу мережі, де всі шари навчаються разом для оптимізації остаточної цілі [12].

2.3 Оптимізатори

Оптимізаційні алгоритми покращують продуктивність моделей глибокого навчання, значно підвищуючи швидкодію та точність тренування моделі. Під час тренування моделі важливим є мінімізування функції втрати та зміна вагів на кожній епосі. Оптимізатор – це алгоритм або функція, яка змінює такі атрибути як ваги та швидкість навчання, що призводить до зниження загальної втрати та покращення точності.

2.3.1 Градієнтний спуск

Цей алгоритм оптимізації систематично модифікує значення ваг та досягає локального мінімуму за допомогою математичних обчислень. Формула (2.5) описує те, що робить алгоритм градієнтного спуску [13].

$$b = \alpha - \gamma \nabla f(\alpha) \tag{2.5}$$

b – наступна позиція, а α – поточна позиція. Мінус перед знаком відповідає мінімізації. Гама посередині – це коефіцієнт ваги, а градієнтна частина – просто напрямок найшвидшого спуску. Таким чином, ця формула показує нам наступну позицію, яка вказує напрямок найшвидшого спуску.

Градієнтний спуск починається з деяких коефіцієнтів, обчислює їх вартість та шукає меншу, ніж наразі. Потім рухається в напрямку меншої ваги і оновлює значення коефіцієнтів. Процес повторюється, поки не буде досягнуто локального мінімуму. Локальний мінімум – це точка, за яку неможливо продовжити рух [14].

Градієнтний спуск є гарним підходом для більшості завдань, але він також має свої недоліки. Якщо розмір даних великий, обчислення градієнту можуть бути часо- та ресурсозатратними. Для конвексних функцій (може мати лише один локальний екстремум (максимум або мінімум) або не мати жодного екстремуму) цей метод працює дуже добре, в той час як для неконвексних він не знає, наскільки далеко рухатися вздовж градієнту і, відповідно, є поганим рішенням.

2.3.2 Стохастичний градієнтний спуск

Як зазначалось раніше, градієнтний спуск на великих наборах даних може не бути найкращим варіантом оптимізатора. Тому був створений стохастичний градієнтний спуск. Цей алгоритм базується на випадковості, що і означає термін «стохастичний». Замість вибору всіх даних ми випадковим чином вибираємо частину даних, яку називають пакетом (batch).

$$w = w - \eta \nabla Q_i(w) \tag{2.6}$$

Формула (2.6) описує кроки, які виконуються в стохастичному градієнтному спуску (SGD). Спочатку ми ініціалізуємо параметри (ваги), позначені як w, і швидкість навчання, позначену як η. Потім на кожній ітерації дані випадковим чином перемішуються, щоб досягти приблизної мінімальної точки [14].

Оскільки ми використовуємо лише частину даних у кожній ітерації, шлях, яким рухається алгоритм, містить багато шуму порівняно з алгоритмом градієнтного спуску. Тому SGD потребує більшої кількості ітерацій, щоб досягти локального мінімуму. Зі збільшенням числа ітерацій збільшується і загальний час обчислення. Проте, навіть після збільшення числа ітерацій, ресурсозатратність все ще нижча, ніж у оптимізатора градієнтного спуску.

2.3.3 Adagrad (Адаптивний градієнтний спуск)

Алгоритм адаптивного градієнтного спуску відрізняється від інших алгоритмів градієнтного спуску тим, що використовує різні швидкості навчання для кожної ітерації. Зміна швидкості навчання залежить від різниці у параметрах під час тренування. Ця модифікація є дуже корисною, оскільки реальні набори даних містять як розріджені, так і щільні ознаки, і недопустимо мати одне значення швидкості навчання для всіх ознак.

$$W_t = W_{t-1} - \eta'_t \frac{\partial L}{\partial w(t-1)} \tag{2.7}$$

$$\eta'_t = \frac{\eta}{\sqrt{\alpha_t + \epsilon}} \tag{2.8}$$

Формули (2.7) і (2.8) показують, як алгоритм Adagrad оновлює ваги. Тут α_t позначає різні швидкості навчання на кожній ітерації, η – це константа, а ϵ – мала позитивна величина, що додається для уникнення ділення на 0 [14].

Використання оптимізатора Adagrad має важливу особливість – немає потреби вручну налаштовувати швидкість навчання. Він збігається до мінімуму швидше і є надійнішим порівняно з алгоритмами градієнтного спуску та іншими варіаціями.

Основним недоліком оптимізатора Adagrad є вірогідність того, що на певному етапі швидкість навчання стає дуже мала, що погіршує точність моделі, оскільки вона не може набути нових знань. Це може статися через накопичення квадратів градієнтів у знаменнику, що призводить до зростання значення знаменника [14].

2.3.4 RMS Prop (Roor Mean Square)

Алгоритм RMS prop є розширеною версією алгоритму RPPROP [15], який вирішує проблему змінних градієнтів. Проблема полягає в тому, що деякі градієнти можуть бути великими, тоді як інші – малими. Тому визначення однієї швидкості навчання може бути не найкращим варіантом. У алгоритмі RPPROP спочатку порівнюються два градієнти за їхніми знаками. Якщо вони мають однакові знаки, то крок збільшується на невелику частку, оскільки ми рухаємося у правильному напрямку. Якщо вони мають протилежні знаки, то крок зменшується. Крок потім обмежується, і ваги оновлюються відповідно.

Однак проблема з алгоритмом RPPROP полягає в тому, що він має складності у роботі з великими наборами даних. Тому був розроблений алгоритм RMS prop, який іноді вважається поліпшенням оптимізатора AdaGrad, оскільки зменшує монотонно спадаючу швидкість навчання [14].

Основний фокус алгоритму RMS prop – прискорити процес оптимізації, зменшивши кількість обчислень функцій, необхідних для досягнення локального мінімуму. Алгоритм зберігає крайнє значення квадратів градієнтів для кожної ваги і ділить градієнт на корінь середньоквадратичного значення. Формула (2.9) математично описує цей процес.

$$v(w,t) = \gamma v(w,t-1) + (1-\gamma)(\nabla Q_i(w))^2 \quad (2.9)$$

Де γ – це коефіцієнт забуття. Ваги оновлюються за формулою (2.10).

$$w = w - \frac{\eta}{\sqrt{v(w,t)}} \nabla Q_i(w) \quad (2.10)$$

Якщо існує параметр, через який функція витрат сильно коливається, ми хочемо уникнути оновлення цього параметра. Цей алгоритм має кілька переваг порівняно з попередніми версіями алгоритмів градієнтного спуску. Він швидко збігається і вимагає менше налаштувань, ніж алгоритми градієнтного спуску та його варіації.

Проблема з RMS Prop полягає в тому, що коефіцієнт навчання має бути визначений вручну, і запропоноване значення не підходить для кожної задачі [14].

2.3.5 Adam

Назва Adam походить від «адаптивної оцінки моменту». Цей алгоритм оптимізації є модифікацією стохастичного градієнтного спуску, що використовується для оновлення ваг мережі під час навчання.

На відміну від стохастичного градієнтного спуску, Adam оновлює коефіцієнт навчання для кожної ваги окремо. Adam також включає переваги алгоритмів AdaGrad і RMSProp. RMSProp адаптує коефіцієнти навчання на основі середньоквадратичного градієнту, тоді як Adam використовує математичне очікування квадратичних відхилень від середнього (яке також називається некоректованою дисперсією).

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\left[\frac{\delta L}{\delta w_t}\right] \quad (2.11)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\left[\frac{\delta L}{\delta w_t}\right]^2 \quad (2.12)$$

Формули (2.11) та (2.12) представляють роботу оптимізатора Adam. Тут β_1 та β_2 відображають частку затухання середнього градієнту [14].

Оптимізатор Adam має декілька переваг, що робить його широко використовуваним та рекомендованим як алгоритм оптимізації за замовчуванням. Він простий у реалізації, має швидший час роботи, вимагає низького обсягу ресурсів та потребує меншого налаштування, ніж інші алгоритми оптимізації.

Однак, навіть у Adam є деякі недоліки. Він схильний до більш швидкого часу обчислення, тоді як алгоритми, такі як стохастичний градієнтний спуск, фокусуються на даних. Тому алгоритми, подібні до SGD, краще узагальнюють дані за рахунок низької швидкості обчислень [14].

2.4 Preprocessing Image Data

Попередня обробка зображень допомагає покращити якість вхідних даних та може призвести до кращої продуктивності моделі. Попередня обробка зображень може допомогти усунути небажані спотворення та покращити конкретні якості, які є важливими для розглянутої задачі. Наприклад, зміна розміру зображень до одного розміру та нормалізація значень пікселів можуть допомогти забезпечити однорідність вхідних даних до нейромережі та використання спільного діапазону значень пікселів [18].

Техніки попередньої обробки, такі як зміна розміру та аугментація даних, можуть допомогти зменшити кількість зображень, які потрібно обробити, і прискорити час навчання моделі без значного впливу на її продуктивність [18][16].

Попередня обробка зображень може допомогти покращити продуктивність моделі, зменшуючи вплив варіацій освітлення, покращуючи контраст та зменшуючи шум на зображеннях [18][17]. Крім того, техніки, такі як передача навчання, можуть використовувати попередньо навчені моделі CNN для поліпшення продуктивності моделі на меншому наборі даних [16].

Існує кілька загальновідомих технік, які можна використовувати. Використання тих чи інших методів залежить від задачі та від вхідних даних.

2.4.1 Centering

Центрування полягає у відніманні середнього значення пікселя від кожного пікселя. Це може допомогти зменшити вплив варіацій освітлення на зображеннях [19].

2.4.2 Standartization

Стандартизація включає віднімання середнього значення пікселя і поділ на стандартне відхилення. Це може допомогти забезпечити схожий масштаб зображення і розподіл значень пікселів .

2.4.3 Histogram equalization

Гістограмне вирівнювання є технікою, яка використовується для покращення контрасту зображення. Вона включає перерозподіл значень інтенсивності пікселів таким чином, щоб вони рівномірніше розподілялися по всьому зображенню.

2.4.4 Batch normalization

Нормалізація пакетів – це техніка, яка використовується для покращення стабільності нейромережі під час навчання. Вона передбачає нормалізацію активацій кожного шару в мережі на основі середнього значення та дисперсії активацій у поточному пакеті [17].

2.4.5 Data augmentation

Аугментація даних – це техніка, яка використовується для штучного збільшення розміру навчального набору даних за рахунок перетворень зображень. Це може допомогти покращити продуктивність моделі та зменшити перенавчання. Розповсюджені методи аугментації даних включають перевертання, обертання, обрізання та масштабування [19].

2.4.6 Transfer learning

Передача навчання – це техніка, що використовується для використання попередньо навчених моделей CNN на великих наборах даних для покращення продуктивності моделі на меншому наборі даних.

2.5 Metrics

Метрики використовуються для оцінки продуктивності CNN. Вибір метриків залежить від конкретної задачі та вимог застосування [20]. Деякі загальноприйняті метрики, які використовуються для оцінки продуктивності мережі:

- Accuracy – відношення правильно класифікованих зразків до загальної кількості зразків.
- Loss – оцінює відповідність між прогнозованою ймовірністю кожного класу та фактичною міткою класу. Чим менша втрата, тим краще модель прогнозує класи.

2.6 Існуючі архітектури CNN

Архітектури CNN постійно розвиваються, оскільки науковці досліджують різні варіанти дизайну та техніки. Дослідники постійно працюють над оптимізацією архітектур для покращення точності, зменшення перенавчання, прискорення навчання, а також мінімізації обчислювальних ресурсів.

Попередньо навчені CNN-моделі часто використовуються як основа для передачі навчання. За допомогою наявних архітектур дослідники та практики можуть налаштовувати моделі на нових наборах даних з обмеженою кількістю анотованих даних, що дозволяє зменшити необхідність у навчанні з нуля.

У цій роботі буде розглянуто 3 існуючі архітектури згорткових мереж:

- ResNet-50
- Inception-v1
- VGG-16

Дані архітектури знаходяться у списку найпопулярніших та найбільш вживаних.

На рисунку 2.10 наведені умовні позначення для компонентів архітектур, що розглядаються.

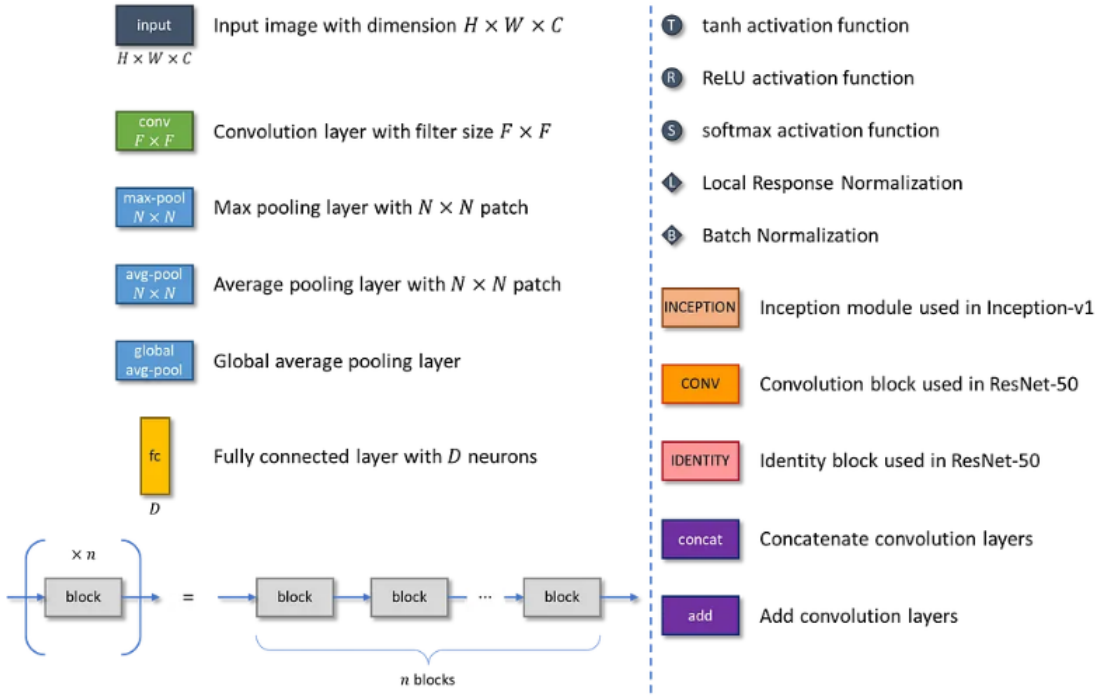

Рисунок 2.10 – Умовні позначення шарів моделі[21]

2.6.1 ResNet-50

При роботі з глибокими згортковими нейронними мережами фахівці з машинного навчання додають більше шарів. Ці додаткові шари допомагають

ефективніше вирішувати складні задачі, оскільки різні шари можуть навчатися для виконання різних завдань та отримувати високоточні результати.

Хоча збільшення кількості шарів може збагатити особливості моделі, глибша мережа може показати проблему деградації. Іншими словами, зі зростанням кількості шарів нейронної мережі рівень точності може насичуватися й поступово знижуватися після певної точки. У результаті продуктивність моделі погіршується як на тренувальних, так і на тестових даних [22].

Ця деградація не є наслідком перенавчання. Замість цього, вона може бути результатом ініціалізації мережі, функції оптимізації або, що є найважливішим, проблеми зникнення або вибуху градієнтів.

Проблема зникнення або вибуху градієнтів виникає, коли градієнти, які передаються зворотньо в мережі під час навчання, стають дуже малими або дуже великими. Це може стати перешкодою для ефективного навчання глибоких мереж, оскільки малі градієнти призводять до повільної збіжності або стагнації навчання, а великі градієнти можуть призвести до нестабільності та непередбачуваної поведінки мережі [22].

Проблема деградації вирішується за допомогою введення блоків згорткових залишків. Існує два типи блоків згорткових залишків (рис. 2.11):

- Блок ідентичності – складається з трьох шарів згортки з розмірами ядра 1×1, 3×3 і 1×1. Ці шари пов’язані між собою за допомогою шарів нормалізації пакетів (Batch Normalization). Функція активації ReLU застосовується до перших двох шарів, тоді як вхід до блоку ідентичності додається до останнього шару перед застосуванням ReLU [21].
- Блок згортки – аналогічний блоку ідентичності, але вхід до блоку згортки спочатку проходить через шар згортки з розміром ядра 1×1 і BN, перед тим як додавати його до останнього шару згортки головної послідовності [21].

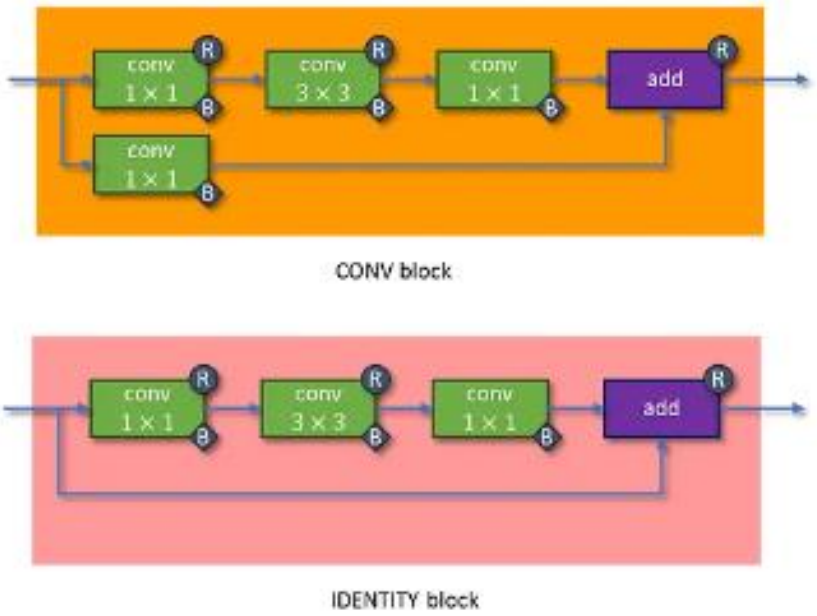

Рисунок 2.11 – Блоки ідентичності та згортки в ResNet-50 [21]

Обидва блоки згорткових залишків мають по 3 шари. Усього, ResNet-50 має 26 мільйонів параметрів і 50 шарів (рис. 2.12) [21]:

- 1 шар згортки з BN, потім застосовується ReLU, за яким слідує
- 9 шарів, які складаються з 1 блоку згортки і 2 блоків ідентичності, за якими слідує
- 12 шарів, які складаються з 1 блоку згортки і 3 блоків ідентичності, за якими слідує
- 18 шарів, які складаються з 1 блоку згортки і 5 блоків ідентичності, за якими слідує
- 9 шарів, які складаються з 1 блоку згортки і 2 блоків ідентичності, за якими слідує
- 1 повнозв’язаний шар з функцією активації softmax.

Перший шар згортки слідує за згорткою розміром 3×3 зі стратегією макс-пулінгу, а останній блок ідентичності слідує за глобальним середнім пулінгом. За замовчуванням, ResNet-50 приймає кольорові зображення розміром 224×224 та видає один з 1000 класів [21].

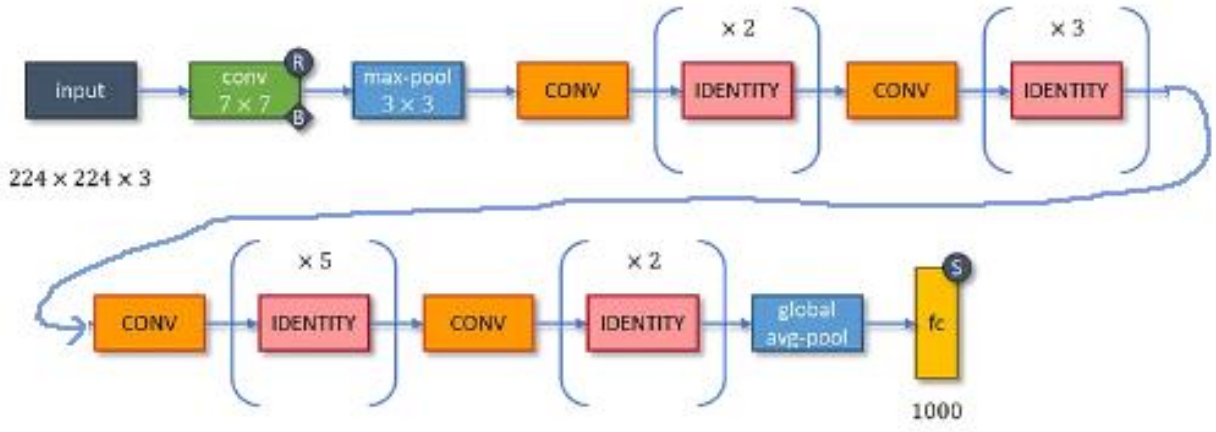

Рисунок 2.12 – Архітектура ResNet-50 [21]

2.6.2 Inception-v1

У конволюційних нейромережах, що існували до появи InceptionNet, основна увага зосереджувалася на збільшенні глибини мережі для витягнення ознак з метою поліпшення здатності моделі до навчання.

Розробники InceptionNet першими зосередились на одночасному збільшенні ширини і глибини моделі, щоб досягти кращої точності при збереженні обчислювальних ресурсів на постійному рівні. Архітектура InceptionNet фокусується на паралельній обробці і одночасному витягненні різних карт ознак. Це основна особливість InceptionNet, яка відрізняє його від інших моделей класифікації зображень [23].

Inception-v1 вирішує проблему вибуху градієнту (опис цього явища наводився у пункті 2.6.1), додаючи два додаткові класифікатори, підключені до проміжних шарів, з метою збільшення сигналу градієнта, який повертається назад. Під час навчання їх втрати додаються до загальної втрати мережі зі знижувальним коефіцієнтом 0.3. Під час інференсу ці допоміжні мережі відкидаються [21].

Inception-v1 вводить модуль Inception (рис. 2.13), який складається з чотирьох серій одного або двох шарів згортки та макс-пулінгу, які розміщуються паралельно і з’єднується в кінці. Модуль Inception має на меті наблизити

оптимальну локальну розріджену структуру в згортковій нейромережі, дозволяючи використовувати кілька типів розмірів ядер, а не обмежуватися одним [21].

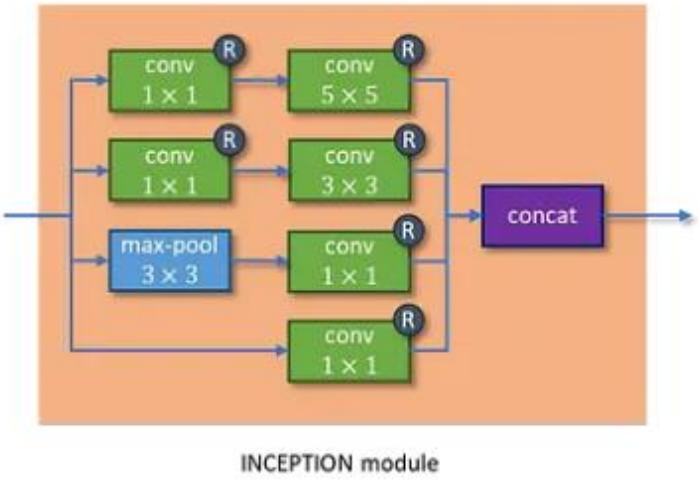

Рисунок 2.13 – Модуль Inception [21]

Inception-v1 має в наявності 22 шари (рис. 2.14) [21]:

- 3 шари згортки з розмірами ядра 7×7, 1×1 і 3×3, за якими слідують
- 18 шарів, які складаються з 9 модулів Inception, кожен з яких має 2 шари згортки/макс-пулінгу, за якими слідує
- 1 повнозв’язаний шар.

Останній шар основного класифікатора та два допоміжних класифікатора оснащені функцією активації softmax, а всі інші мають функцію активації ReLU. 1-й та 3-й шари згортки, а також 2-й та 7-й модулі Inception слідують за макс-пулінгом розміром 3×3. Останній модуль Inception слідує за середнім пулінгом розміром 7×7. Після 1-го макс-пулінгу та 3-го шару згортки застосовується LRN (локальна нормалізація розміром).

Додаткові класифікатори розгалужуються після 3-го та 6-го модулів Inception. Кожен з них починається з середнього пулінгу розміром 5×5 і потім має 3 шари [21]:

- 1 шар згортки з розміром ядра 1×1, а потім
- 2 повнозв’язаних шари.

За замовчуванням Inception-v1 приймає зображення з кольоровими просторовими розмірами 224×224 і виводить один з 1000 класів.

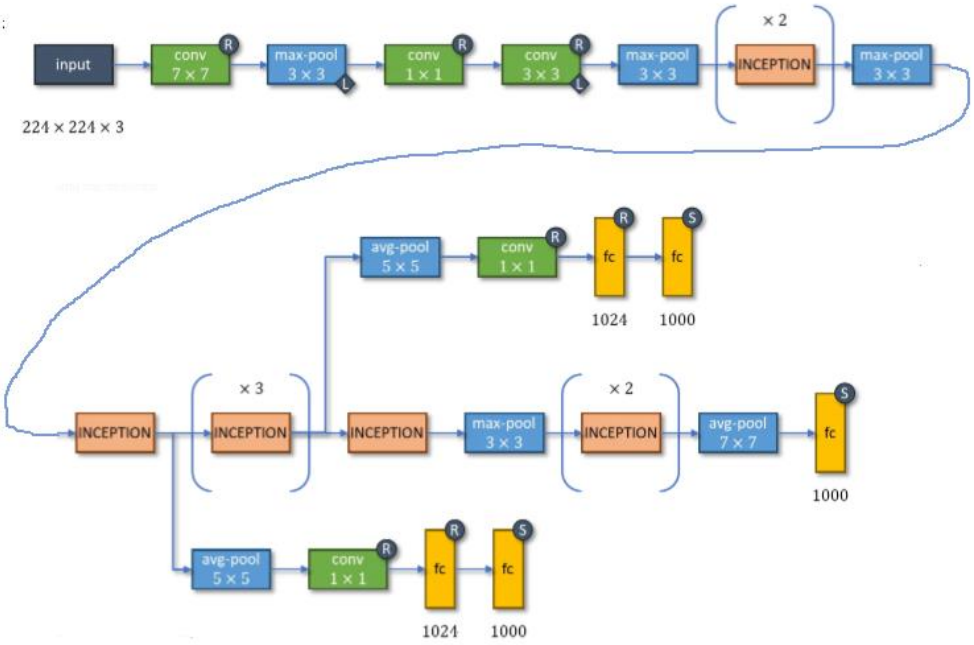

Рисунок 2.14 – Архітектура Inception-v1 [21]

2.6.3 VGG-16

VGG16 вважається однією з найкращих моделей комп'ютерного зору на сьогоднішній день. Творці цієї моделі оцінили різні конфігурації мережі і збільшили її глибину, використовуючи архітектуру з дуже малими (3 × 3) фільтрами згортки, що привело до значного покращення в порівнянні з попередніми реалізаціями. Вони збільшили глибину до 16–19 вагових шарів, що приблизно становить 138 мільйонів параметрів для навчання.

VGG16 може класифікувати 1000 зображень з 1000 різних категорій з точністю 92,7% [24].

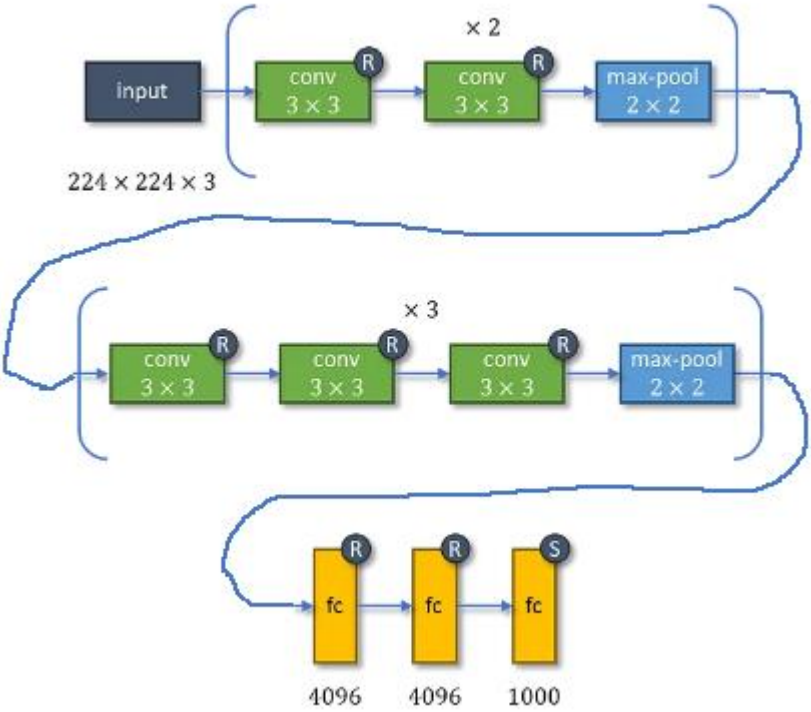

Рисунок 2.15 – Архітектура VGG-16 [21]

VGG-16 складається із 16 шарів (рис. 2.15) [21]:

- 13 шарів згортки з розміром ядра 3×3, а потім
- 3 повнозв’язаних шари.

Останній шар оснащений функцією активації softmax, а всі інші шари мають функцію активації ReLU. Після 2-го, 4-го, 7-го, 10-го та 13-го шарів згортки слідує пулінг розміром 2×2. За замовчуванням VGG-16 приймає зображення з кольоровими просторовими розмірами 224×224 і виводить однин з 1000 класів [21].

2.7 Опис власної архітектури

Ретельно дослідивши існуючі архітектури ResNet, Inception та VGG, створено власну модель, що складається з п’яти блоків Conv, що містять поєднання шарів SeparableConv2D, BatchNormalization, MaxPooling та Dropout. Вихід останнього блоку Conv згладжується і за ним слідують три повністю з’єднані (FC) шари, кожен зі своїм власним шаром Dropout. Додається останній

FC шар з чотирма нейронами (відповідає кількості класів лейкоцитів) і функцією активації softmax для багатокласової класифікації.

Розлянемо детальніше створену архітектуру (рис. 2.16):

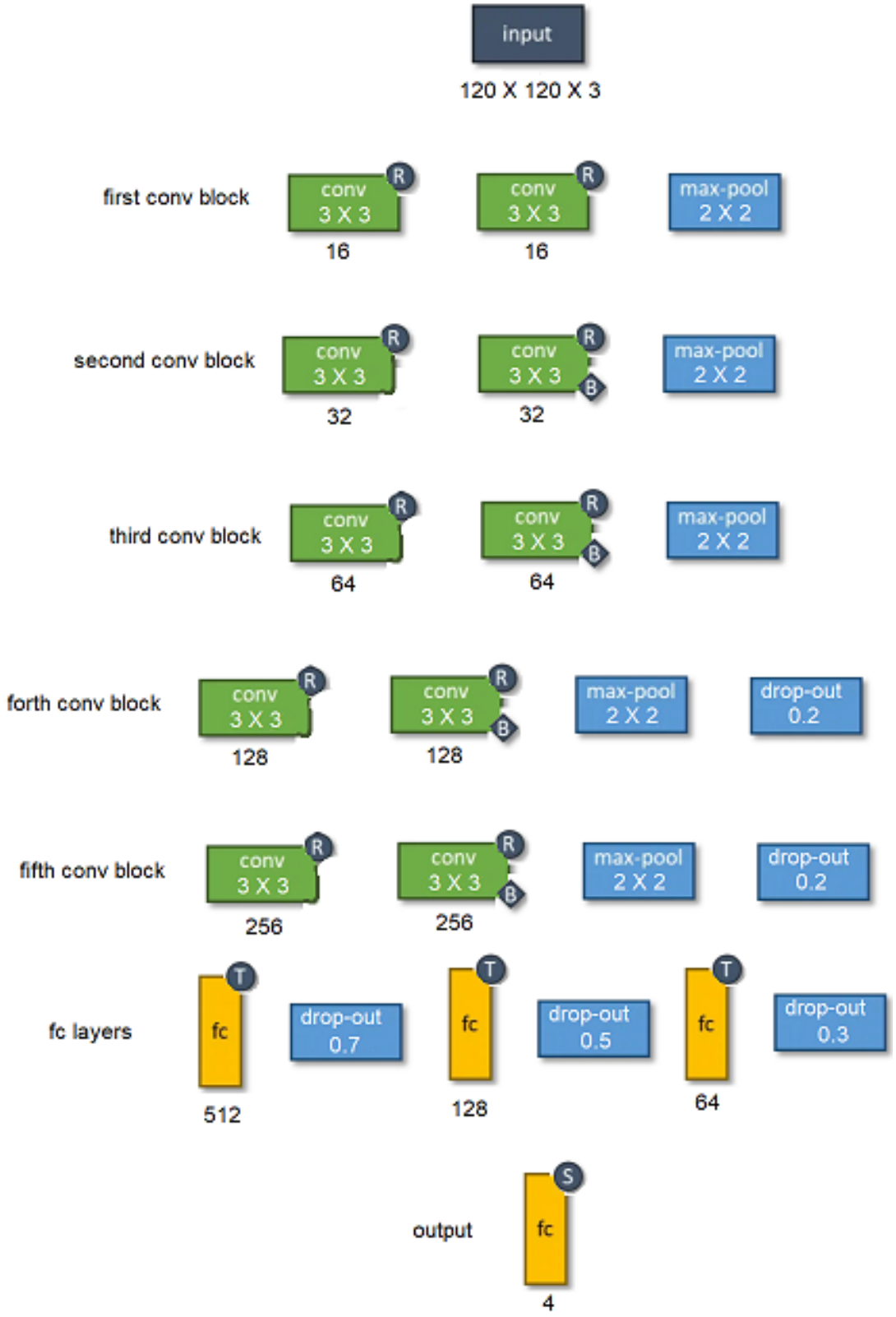

Рисунок 2.16 – Архітектура створеної моделі

- Input layer

  Розміри зображень, що подаються на вході дорівнюють 120 x 120 x 3.

- Перший блок.

  Два згорткових шари для вилучення ознак з вхідних зображень. Кожен згортковий шар має 16 фільтрів розміром 3x3. Застосовується активаційна функція ‘relu’, що вводить нелінійність. Параметр padding встановлено на ‘same’, щоб зберегти просторові розміри вхідних та вихідних ознакових карт. Padding допомагає зберегти інформацію на краях ознакових карт. Використовується шар MaxPooling2D з розміром пулінгу 2x2, щоб зменшити розмір ознакових карт.

- Другий блок.

  Два роздільні згорткові шари для захоплення більш складних ознак. Кожен роздільний згортковий шар має 32 фільтри розміром 3x3. Застосовується активаційна функція ‘relu’, що вводить нелінійність. Різновидні згорткові шари використовують комбінацію глибинного та точкового згорткових шарів, що допомагає зменшити кількість параметрів та обчислень. Після роздільних згорткових шарів додається шар BatchNormalization для нормалізації активацій, що покращує стабільність та швидкість навчання. Використовується шар MaxPooling2D для зменшення розміру.

- Третій блок.

  Два роздільні згорткові шари з 64 фільтрами розміром 3x3. Аналогічно до попередніх блоків, застосовується активаційна функція ‘relu’ та шар BatchNormalization. Використовується шар MaxPooling2D для зменшення розміру.

- Четвертий блок.

Два роздільні згорткові шари з 128 фільтрами розміром 3x3. Застосовується активаційна функція ‘relu’ та шар BatchNormalization. Використовується шар MaxPooling2D для зменшення розміру. Додається шар Dropout зі значенням 0.2 для запобігання перенавчанню. Dropout випадково встановлює певну частку вхідних нейронів на значення 0 під час навчання, що допомагає уникнути занадто сильної залежності моделі від окремих ознак.

- П’ятий блок.

  Два роздільні згорткові шари з 256 фільтрами розміром 3x3. Застосовується активаційна функція ‘relu’ та шар BatchNormalization. Використовується шар MaxPooling2D для зменшення розміру. Додається шар Dropout зі значенням 0.2.

- Повністю з’єднані шари (FC layers).

  Ознакові карти перетворюються в одновимірний вектор для подальшої передачі у повністю з’єднані шари (Flattered). Додається повністю з’єднаний шар з 512 вузлами та активаційною функцією ‘tanh’. Додається шар Dropout зі значенням 0.7. Додається повністю з’єднаний шар з 128 вузлами та активаційною функцією ‘tanh’. Додається шар Dropout зі значенням 0.5. Додається повністю з’єднаний шар з 64 вузлами та активаційною функцією ‘tanh’. Додається шар Dropout зі значенням 0.3.

- Output layer.

  Повністю з’єднаний шар (FC) з 4 вузлами (відповідає кількості класів лейкоцитів) та активаційною функцією ‘softmax’. Функція ‘softmax’ використовується для отримання ймовірностної розподілу для кожного класу.

Як результат, створена модель містить 1 397 028 параметрів. Кількість тренувальних параметрів дорівнює 1 396 068, а нетренувальних – 960.

2.8 Висновки до розділу 2

У другому розділі ретельно досліджено основи згорткових нейронних мереж, які широко використовуються для класифікації зображень. Також розглянуто як комп’ютер інтерпретує вхідні зображення.

Проаналізувано базові поняття архітектури згорткових мереж, такі як шари згортки, об’єднання, функції активації, повнозв’язні шари.

Розглянуто різні оптимізатори Adam, Adagrad, RMS Prop, Gradient Descent, SGD, які широко використовуються в глибинному навчанні.

Також розглянуто основні принципи передобробки зображень (центрування, нормалізація, аугументація, стандартизація), які можна застосовувати при роботі зі згортковими нейронними мережами.

Приділено увагу метрикам (accuracy, precision, f1 score, recall, confusion matrix), які використовуються для оцінки продуктивності мережі в контексті задачі класифікації зображень.

Представили 3 популярних архітектури згорткових нейронних мереж, а саме VGG-16, Inception-v1 та ResNet-50. Кожну з цих архітектур супроводжує зображення з її структурою, а також пояснення найцікавіших та незвичних модулів, завдяки яким ці мережі стали так популярними.

Розроблено власну архітектуру, описано всі її компоненти.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Архітектура системи та вибір мови програмування та фреймворків

Основні вимоги до архітектури були поверхнево наведені у розділі 1.4. Нижче наведені більш широкі вимоги до системи:

- Інтерфейс користувача. Додаток повинен мати зрозумілий та зручний інтерфейс, який дозволяє користувачеві легко взаємодіяти з програмою. Це може включати загрузку зображень крові, відображення результатів класифікації та навігаційні елементи для взаємодії з додатком.
- Завантаження зображень. Додаток повинен мати можливість завантажувати зображення крові, які будуть піддаватися аналізу. Це може бути реалізовано за допомогою кнопки завантаження або перетягування зображень у вікно додатку.
- Візуалізація результатів. Додаток повинен забезпечувати візуалізацію результатів класифікації, наприклад, шляхом виділення лейкоцитів на зображенні або виведення результатів у вигляді графіків або таблиць.
- Швидкодія. Додаток повинен бути ефективним та швидкодіючим, здатним обробляти зображення крові в реальному часі або в максимально короткий час.
- Надійність. Додаток повинен бути стабільним та надійним, здатним працювати без збоїв протягом тривалого часу. Він повинен враховувати можливі помилки та відновлюватися від непередбачуваних ситуацій.
- Масштабованість. Додаток повинен бути готовим до масштабування, здатним обробляти велику кількість зображень та працювати з багатьма користувачами одночасно.

Робота веб-додатка доволі примітивна: застосунок отримує запит від користувача на стороні клієнту, після чого за допомогою нейронних моделей

виконує обрахунки на сервері, далі формує веб сторінку і відправляє її користувачу.

Тобто в нас є три основні аспекти для розробки – клієнт, сервер та нейронні моделі. Необхідно розглянути кожен детальніше.

3.1.1. Моделі нейронних мереж

Архітектури, що використовуються були описані в 2 розділі. Варто зазначити, що розробка велась на платформі Kaggle. Середовище розробки там нагадує Jupyter Notebook (весь код можна поділити на частини, та окремо модифікувати, запускати та редагувати їх). Також вагомою перевагою є можливість безкоштовного доступу до графічних процесорів, які значно пришвидшать навчання моделей.

Мовою програмування було обрано Python. Python має простий і логічний синтаксис, що полегшує читання і написання коду. Python має велику екосистему бібліотек і фреймворків, спеціально розроблених для машинного навчання і нейронних мереж.

Основою для навчання моделей є фреймворк TensorFlow, але із бібліотек Keras, PyTorch і scikit-learn також використовувались певні функції. TensorFlow доволі гнучкий, дозволяючи будувати та налаштовувати різні типи архітектур нейронних мереж. Він надає розширений набір інструментів та функцій для визначення, навчання та оцінки складних моделей. Фреймворк має такі функції, як автоматична диференціація, яка спрощує процес обчислення градієнтів під час навчання. TensorFlow надає інструменти для візуалізації та налагодження моделей нейронних мереж. Він має вбудовані можливості для відображення обчислювальних графів, моніторингу процесу навчання та внутрішньої структури моделі, що допомагає зрозуміти та усунути проблеми в складних моделях. TensorFlow безперешкодно інтегрується з іншими популярними бібліотеками машинного навчання та обробки даних, такими як NumPy, SciPy та Pandas. Ця інтеграція дозволяє використовувати переваги цих бібліотек разом з

можливостями TensorFlow, покращуючи здатність ефективно обробляти та маніпулювати даними.

3.1.2. Клієнт

Frontend – це клієнтська частина програмного забезпечення, яка виконується в браузері користувача. Вона відповідає за відображення і взаємодію з інтерфейсом. Для створення та функціональності веб-додатків, Frontend використовує JavaScript, а також HTML та CSS. Клієнт забезпечує спілкування з сервером для отримання та обробки данних, а також інтерактивність та навігацію для користувача.

Відповідно мовою програмування frontend-частини було обрано JavaScript.

JavaScript містить велику кількість фреймворків для розробки веб-сторінок. Найпопулярнішими є React, Angular та VueJs. Саме React було обрано для розв’язання поставленої задачі. Реакт базується на компонентній архітектурі, що сприяє повторному використанню коду, легкості управління станом компонентів та швидкому розширенню функціональності. Він має велику та активну спільноту розробників, що забезпечує широку підтримку, документацію та навчальні ресурси. Крім того, існує багато сторонніх бібліотек і інструментів, які роблять розробку з використанням реакту ще більш зручною. Завдяки великій кількості сторонніх бібліотек та плагінів, Реакт може бути легко розширений для задоволення конкретних потреб проекту.

3.1.3. Сервер

Серверна розробка відноситься до створення програмного забезпечення, яке виконується на сервері та взаємодіє з клієнтськими запитами. Серверна частина може включати в себе розробку веб-серверів, баз даних, додаткової логіки та обробки даних, яка відбувається на боці сервера. Серверна розробка є важливим елементом для створення функціональних та ефективних веб-

додатків, які забезпечують обмін даними та функціональність між сервером і клієнтом.

Мовою програмування backend-частини проекту було обрано Python. Перш за все, нейронні моделі були створення також за допомогою Python, відповідно інтеграція моделей із backend-частиною не викличе ніяких проблем. Також для розробки серверної частини веб додатка на python існує достатньо велика кількість фреймворків та бібліотек, які дозволяють легко будувати backend-частину проекту.

Фреймворком було обрано FastAPI. Це сучасний, швидкий та високопродуктивний фреймворк для створення веб-додатків та API на мові програмування Python. FastAPI надає простоту та зрозумілість у розробці завдяки автоматичній генерації документації за допомогою внутрішніх стандартів та підтримці автоматичного створення схеми API. Його особливістю є висока продуктивність та можливість асинхронного виконання, що дозволяє обробляти багато запитів одночасно і забезпечувати швидку відповідь на них. FastAPI є популярним вибором для створення API та веб-додатків у світі Python завдяки своїй продуктивності, простоті використання та багатофункціональності.

3.2. Попередня обробка зображень

Нажаль, набір даних не містить координат лейкоцитів, тому було необхідно використовувати масковий фільтр для їх ізоляції на зображеннях. Як показано на прикладі зображення з набору даних (рис. 3.1), фон (тобто область, яка не є еритроцитом або лейкоцитом) має сірий колір, колір еритроцитів нахиляється до червоних відтінків, а колір лейкоцитів має сильну синю компоненту.

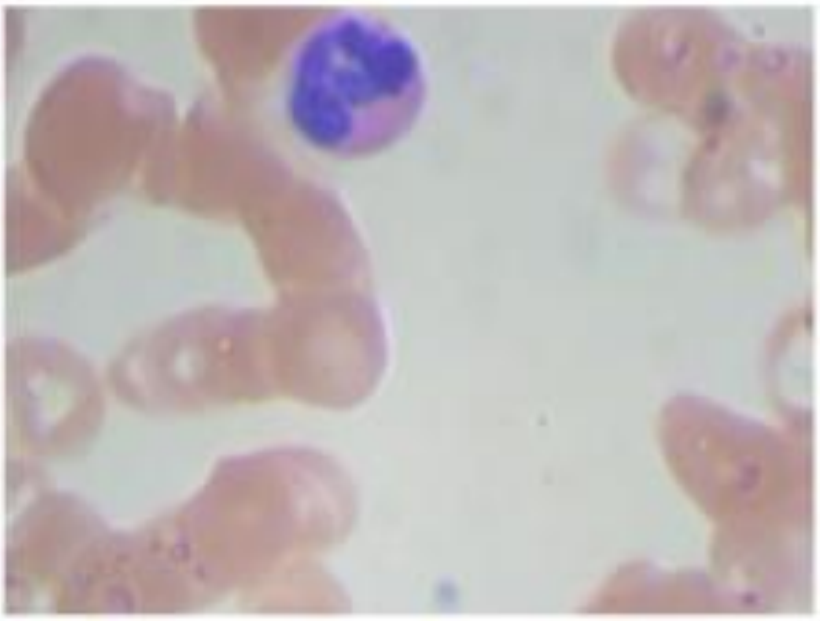

Рисунок 3.1 – Приклад зображення з навчального набору даних

Після застосування маскового фільтра, зображення все ще залишається не дуже «чистим» (рис. 3.2).

Рисунок 3.2 – Зображення з багатьма зайвими фрагментами

Для доповнення маскового фільтру необхідно виконати наступні кроки:

- Позбутися від дрібних залишків та зробити маски більш круглими.
- Знайти огороджуючі прямокутники навколо лейкоцитів.

- Завершити обрізку лейкоцитів відповідно до їх огороджуючих прямокутників.

В результаті було отримано ізольовані зображення лейкоцитів, які використовувалися як приклади для навчання моделі (Рис. 3.3).

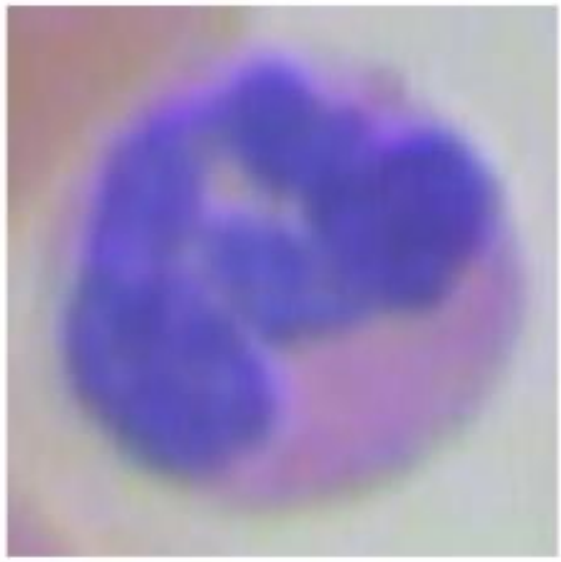

Рисунок 3.3 – Ізольований лейкоцит

3.3. Порівняльний аналіз навчених моделей

Розглянемо результати моделей ResNet50, Inception-v1, VGG-16, а також власної архітектури, а потім наведемо порівняльну таблицю, щоб визначити, яка модель покаже себе краще.

Для моделей на основі ResNet50, VGG-16 та Inception-v1 умови запуску є однаковими:

- Кількість епох – 10
- Розмір батча – 32
- Оптимізатор – Adam
- Learning rate – 10^{-3}

Для кожної архітектури будуть наведені зображення зміни значень accuracy та loss у вигляді графіків. На графіках дві лінії, зелена – зміна значень на тренувальних даних, в той час як червона – на валідаційних.

3.3.1. ResNet-50

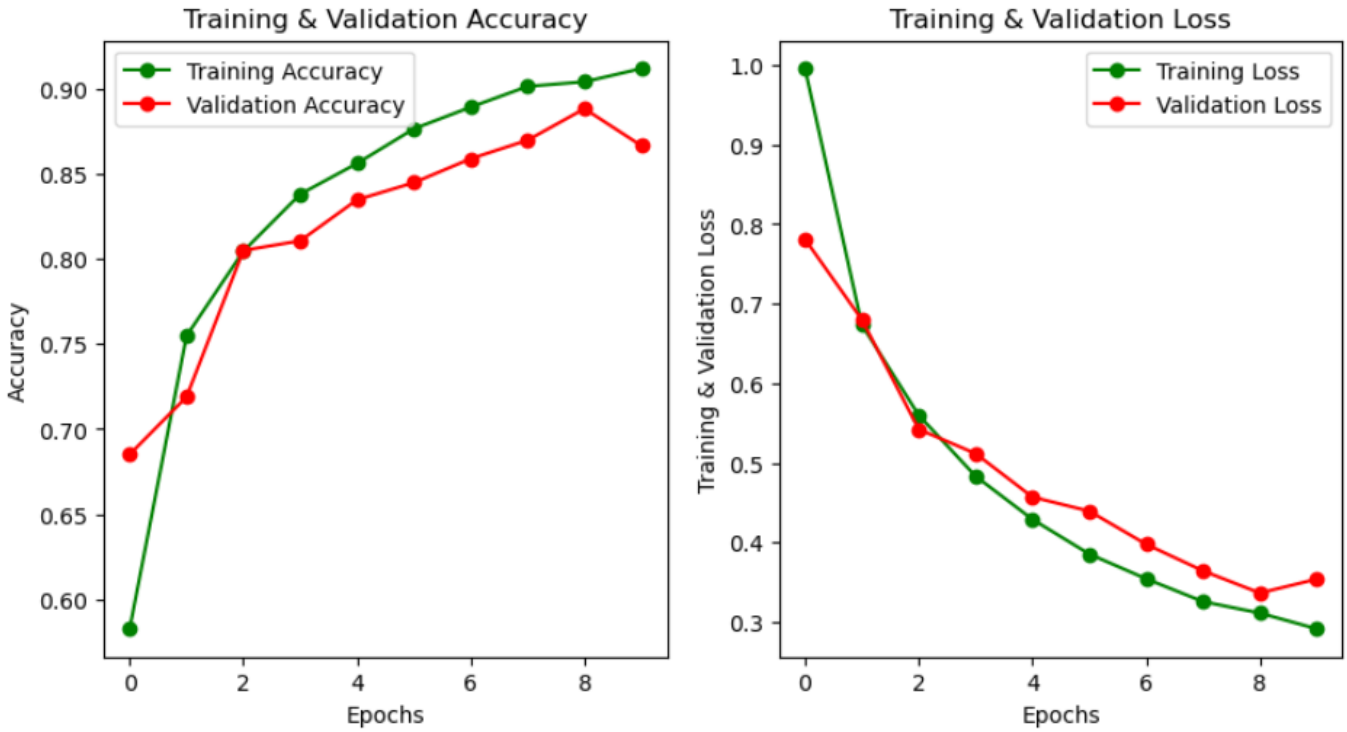

Рисунок 3.4 – Графік навчання моделі на основі ResNet-50

Як видно із рисунку 3.4 на першій епосі модель досягає точності близько 58%, але з кожною наступною епохою точність збільшується, а втрати зменшуються. Після 10 епох точність становить приблизно 91.2%, а втрати складають близько 0.29 на тренувальному наборі даних. На наборі даних перевірки точність становить близько 86.6% з втратами близько 0.35. Це означає, що модель поступово вдосконалюється під час навчання, зменшуючи втрати і підвищуючи точність.

У таблиці 3.1 наведені точні значення accuracy та loss для тренувальних, валідаційних та тестових даних на кінець навчання моделі.

Таблиця 3.1

Результати навчання моделі на основі ResNet-50

	Train	Validation	Test
Accuracy	0.9118	0.8665	0.9276
Loss	0.2917	0.3541	0.2834

3.3.2. Inception-v1

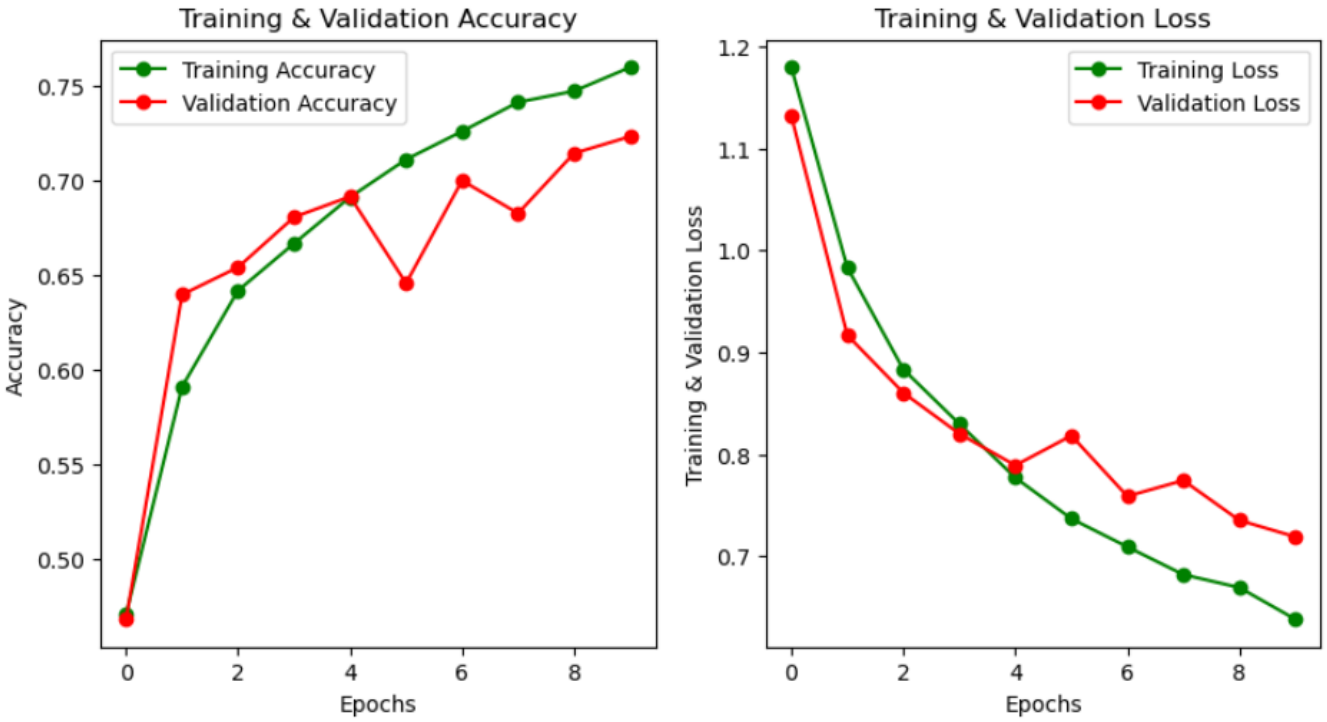

Рисунок 3.5 – Графік навчання моделі на основі Inception-v1

Як видно із рисунку 3.5 на першій епосі модель досягає точності близько 47%, але з кожною наступною епохою точність збільшується, а втрати зменшуються. Після 10 епох точність становить рівно 76%, а втрати складають близько 0.63 на тренувальному наборі даних. На наборі даних перевірки точність становить близько 72% з втратами близько 0.71.

У таблиці 3.2 наведені точні значення accuracy та loss для тренувальних, валідаційних та тестових даних на кінець навчання моделі.

Таблиця 3.2

Результати навчання моделі на основі Inception-v1

	Train	Validation	Test
Accuracy	0.7600	0.7234	0.7823
Loss	0.6381	0.7190	0.5987

3.3.3. VGG-16

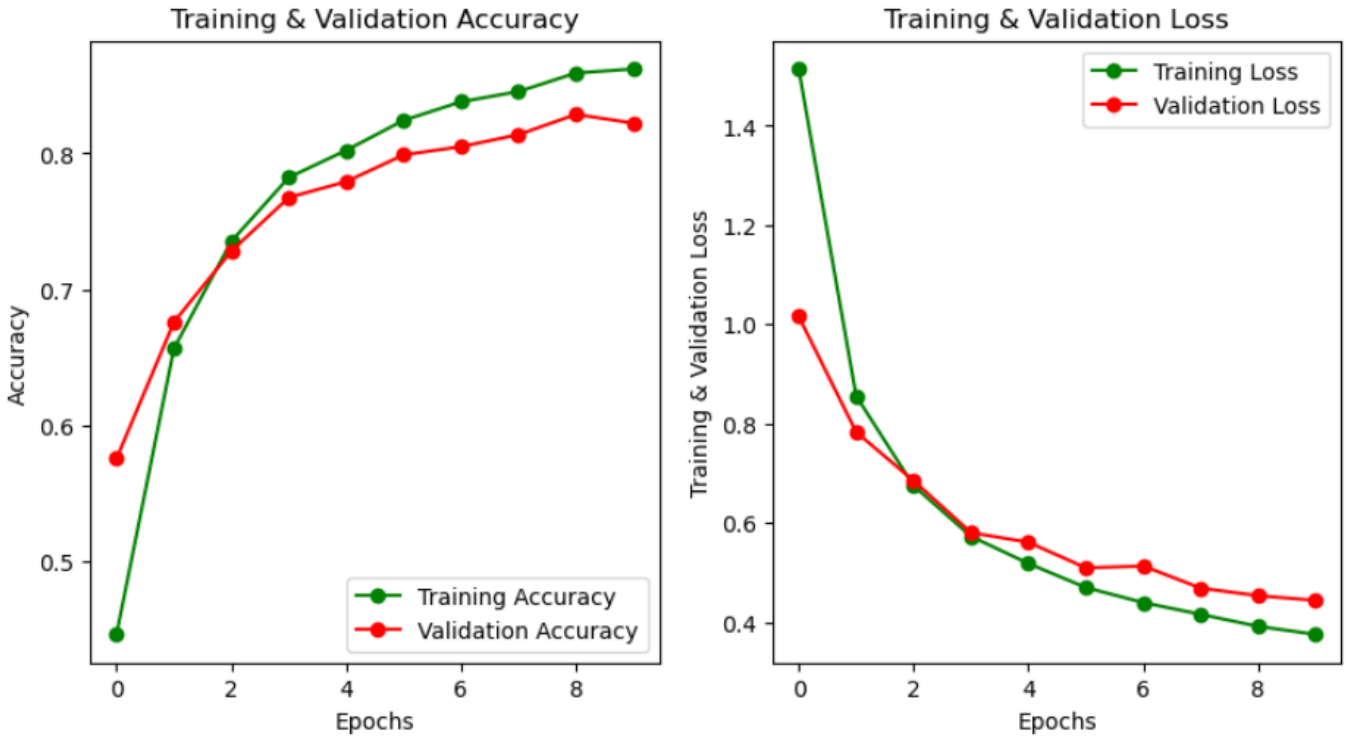

Рисунок 3.6 – Графік навчання моделі на основі VGG-16

Як видно із рисунку 3.6 на першій епосі модель досягає точності близько 44%, але з кожною наступною епохою точність збільшується, а втрати зменшуються. Після 10 епох точність становить близько 86%, а втрати складають близько 0.44 на тренувальному наборі даних. На наборі даних перевірки точність становить близько 82% з втратами близько 0.44.

У таблиці 3.3 наведені точні значення accuracy та loss для тренувальних, валідаційних та тестових даних на кінець навчання моделі.

Таблиця 3.3

Результати навчання моделі на основі VGG-16

	Train	Validation	Test
Accuracy	0.8621	0.8223	0.8856
Loss	0.3752	0.4438	0.3612

3.3.4. Власна модель

Для власної архітектури із умов запуску змінюється тільки кількість епох, тепер вона дорівнює 30.

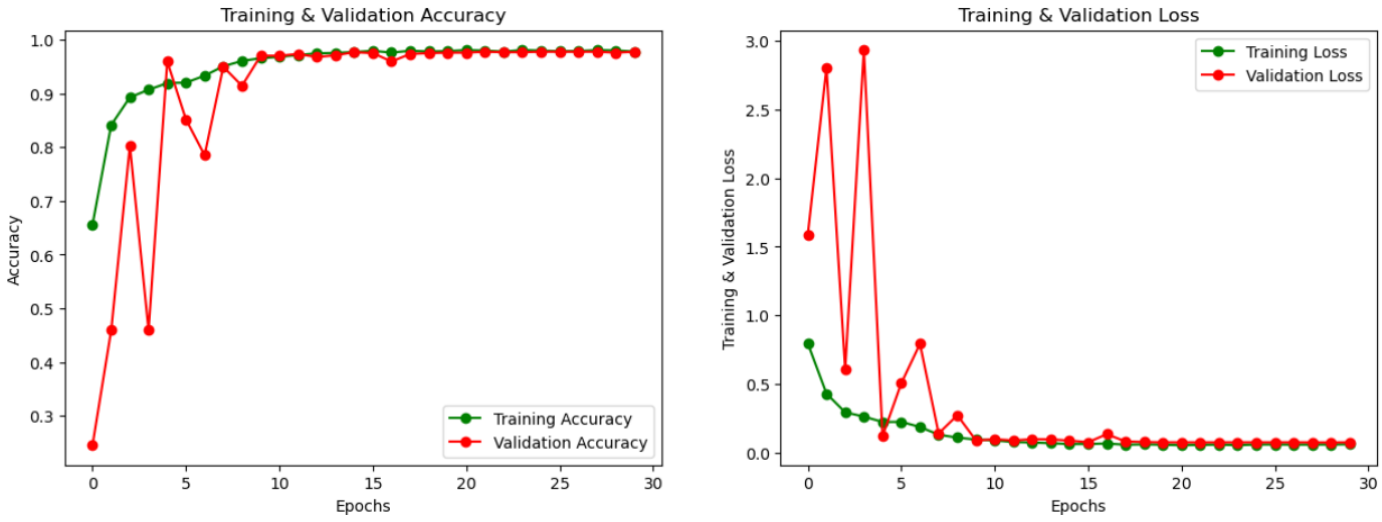

Рисунок 3.7 – Графік навчання власної моделі

Як видно із рисунку 3.7 на першій епосі модель досягає точності близько 67%, але з кожною наступною епохою точність збільшується, а втрати зменшуються. Після 30 епох точність становить близько 97%, а втрати складають близько 0.06 на тренувальному наборі даних. На наборі даних перевірки точність становить близько 97% з втратами близько 0.07.

У таблиці 3.4 наведені точні значення accuracy та loss для тренувальних, валідаційних та тестових даних на кінець навчання моделі.

Таблиця 3.4

Результати навчання власної моделі

	Train	Validation	Test
Accuracy	0.9782	0.9778	0.9801
Loss	0.0613	0.0756	0.0605

3.3.5. Висновок стосовно результатів навчених моделей

Таблиця 3.5

Порівняння архітектур

Архітектура	Accuracy			Loss		
	train	validation	test	train	validation	test
ResNet-50	0.9118	0.8665	0.9276	0.2917	0.3541	0.2834
Inception	0.7600	0.7234	0.7823	0.6381	0.7190	0.5987
VGG-16	0.8621	0.8223	0.8856	0.3752	0.4438	0.3612
Власна модель	0.9782	0.9778	0.9801	0.0613	0.0756	0.0605

Помаранчевим кольором було виділено показники найкращої моделі.

Як видно із таблиці 3.5 власна модель показала найкращі результати серед усіх. Воно і не дивно, адже ця архітектура була створена для вирішення конкретно задачі класифікації лейкоцитів. Для будь якої іншої задачі створена архітектура покаже набагато гірші результати, в той час як ResNet-50, Inception-v1 та VGG-16 були створені як універсальні моделі, які на будь яких задачах класифікації об’єктів на зображеннях показують хороші результати.

3.4. Приклад роботи веб додатку

Користуватись веб додатком дуже просто. Спочатку користувача зустрічає напис та кнопка (рис. 3.8), що пропонують йому обрати зображення.

Рисунок 3.8 – Початковий стан веб сторінки

Далі залишається натиснути кнопку «Classify», після чого на сервер передається запит, а сервер у свою чергу віддає відповідь, яку бачить користувач нижче. Відповідь представлена у вигляді зображення (рис. 3.9), на якому визначено(позначено червоним прямокутником) лейкоцит, а знизу написано його тип.

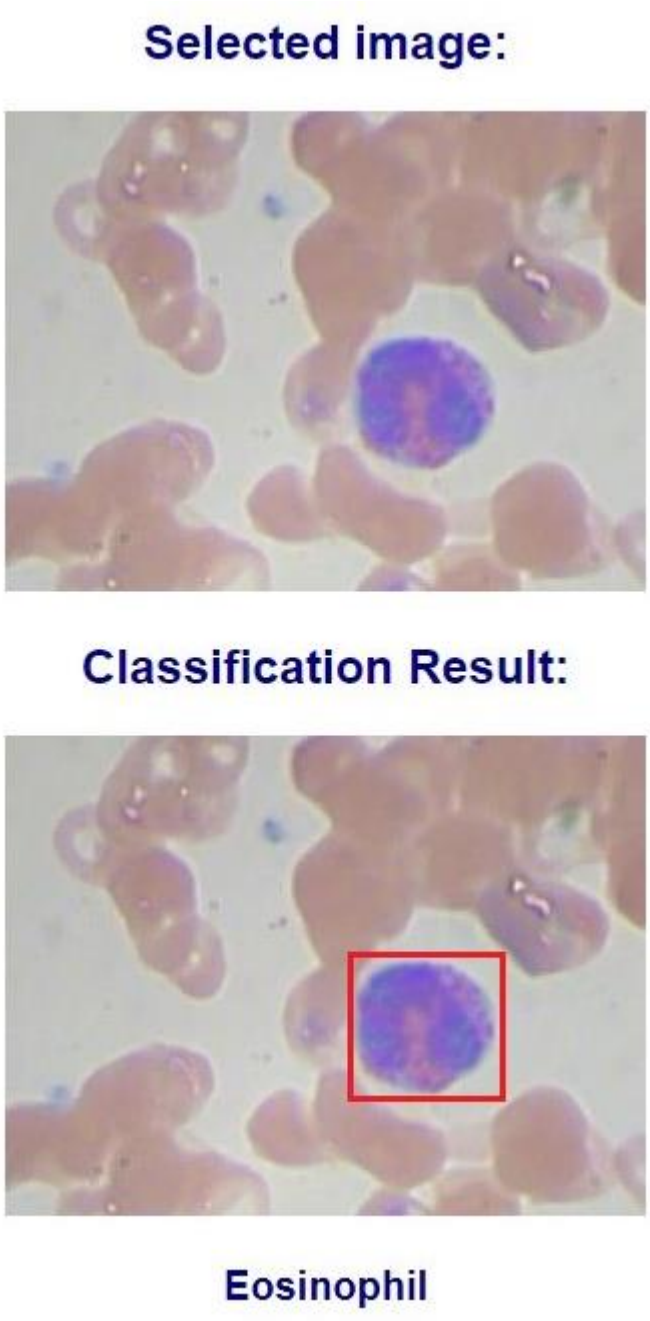

Рисунок 3.9 – Приклад роботи веб-додатку

3.5. Висновки до розділу 3

У даному розділі детальніше описано вимоги до системи, що розроблюється.

Також описано та обгрунтовано вибір мов програмування та фрейморків для розробки компонентів системи. Обрано Python та TensorFlow для нейронних

моделей, JavaScript та React для клієнтської частини, а також Python та FastApi для серверної сторони веб-додатку.

Описано попередню обробку вхідних даних за допомогою маскових фільтрів.

Далі проведено порівняльний аналіз навчених моделей. Наведено графіки змін значень accuracy та loss для тренувальних та валідаційних даних протягом навчання. Також створено таблиці із точними значеннями цих метриків. Із результатів видно, що власна архітектура показала найкращі результати, а отже саме вона буде використовуватися у веб-додатку для класифікації лейкоцитів на зображеннях.

У останньому пункті продемонстровано та описано роботу створеного веб-додатку.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

В заданому розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на розпізнаванні та класифікації лейкоцитів на зображеннях крові, використовуючи глибинне навчання.

Дана реалізація буде сприяти проведенню усіх необхідних досліджень, що дасть змогу якісно дослідити питання не лише в Україні, проте у всьому світі.

Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи класифікації лейкоцитів на зображеннях крові. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її

задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по класифікації.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних 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 та класифікації лейкоцитів на зображеннях крові. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування.

F_2 – вибір фреймворку машинного навчання.

F_3 – вибір середовища програмування.

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) Kaggle (Jupyter Notebook).

б) PyCharm.

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

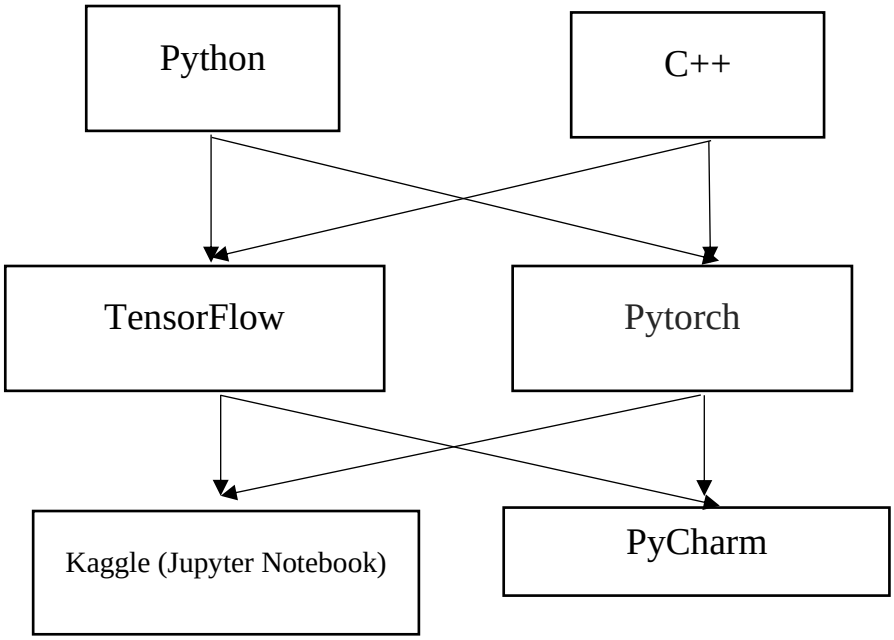

Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1.

Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки

F_1	*А*	Зручність, багатофункціональність, широкий вибір бібліотек	Швидкодія, обмежена підтримка апаратного прискорення
	Б	Висока швидкодія, доступ до оптимізаційних можливостей процесора,	Складність, менша кількість бібліотек, вимога до вміння ручного керування пам'яттю
F_2	*А*	Широкий вибір моделей, розширені можливості управління пам'яттю	Складніший синтаксис, більша кількість коду для досягнення тих самих результатів,.
	Б	Простий синтаксис, динамічні графи, легкість у розробці та налагодженні моделей,	Менша швидкодія порівняно з TensorFlow на деяких завданнях
F_3	*А*	Інтерактивність, можливість візуалізації даних та результатів, зручність	Потребує багато ресурсів, обмежена підтримка великих обсягів даних,
	Б	Підтримка автодоповнення, інструменти для оптимізації, робота з великими проектами	Висока ціна платної версії, проблеми з налаштуванням деяких бібліотек

На основі аналізу позитивно-негативної матриці 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 даємо зручності, багатофункціональності, широкому вибору бібліотек. Для спрощення роботи по написанню коду варіант Б має бути відкинутий.

Функція F_2:

Програма допускає обрання обох варіантів. Можливо використати варіанти А чи Б.

Функція F_3:

Реалізація першого варіанту є сприйнятливою для програми. Це варіант А.

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 програмного продукту

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* – швидкодія мови програмування;
- *X2* – об’єм пам’яті для обчислень та збереження даних;
- *X3* – час навчання даних;
- *X4* – потенційний об’єм програмного коду.

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

Таблиця 4.2

Основні параметри програмного продукту

Назва			Значення параметра

Параметра	Умовні позначення	Одиниці виміру	гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	8000	12000	16000
Об’єм пам’яті	X2	Мб	256	128	64
Час попередньої обробки даних	X3	мс	5	4	3
Потенційний об’єм програмного коду	X4	кількість рядків коду	5000	2500	1000

За даними таблиці 4.3 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

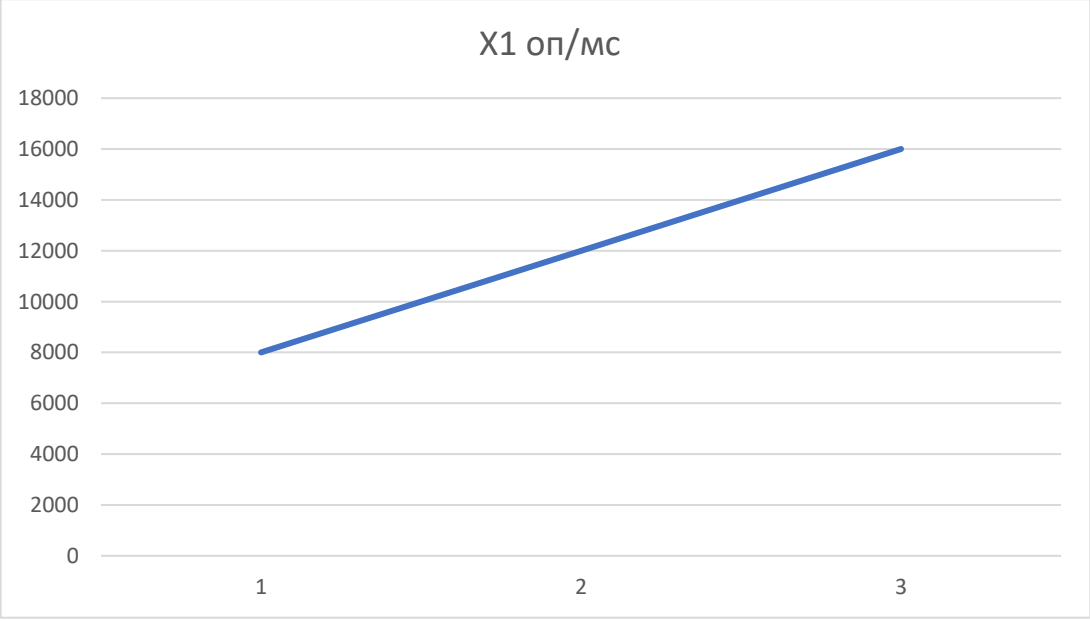

Рисунок 4.2 – X1, швидкодія мови програмування

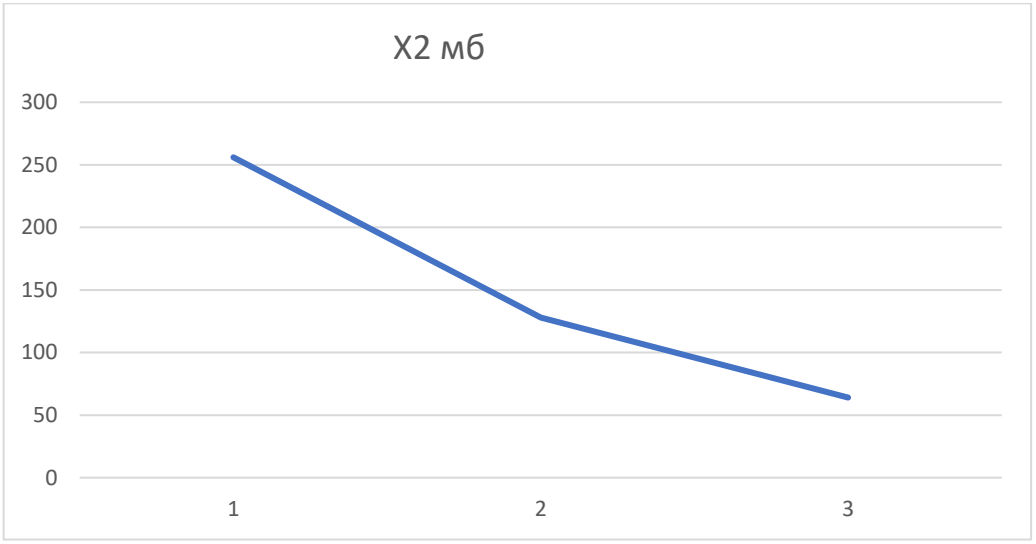

Рисунок 4.3 – X2, об’єм пам’яті

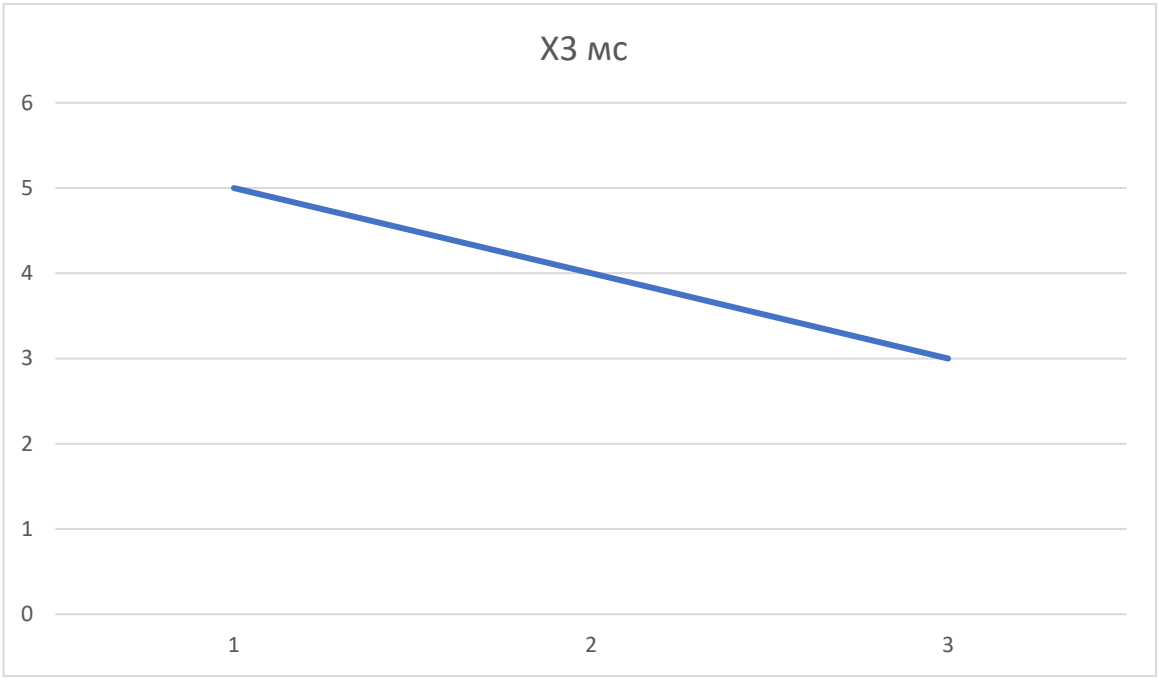

Рисунок 4.4 – X3, час попередньої обробки даних

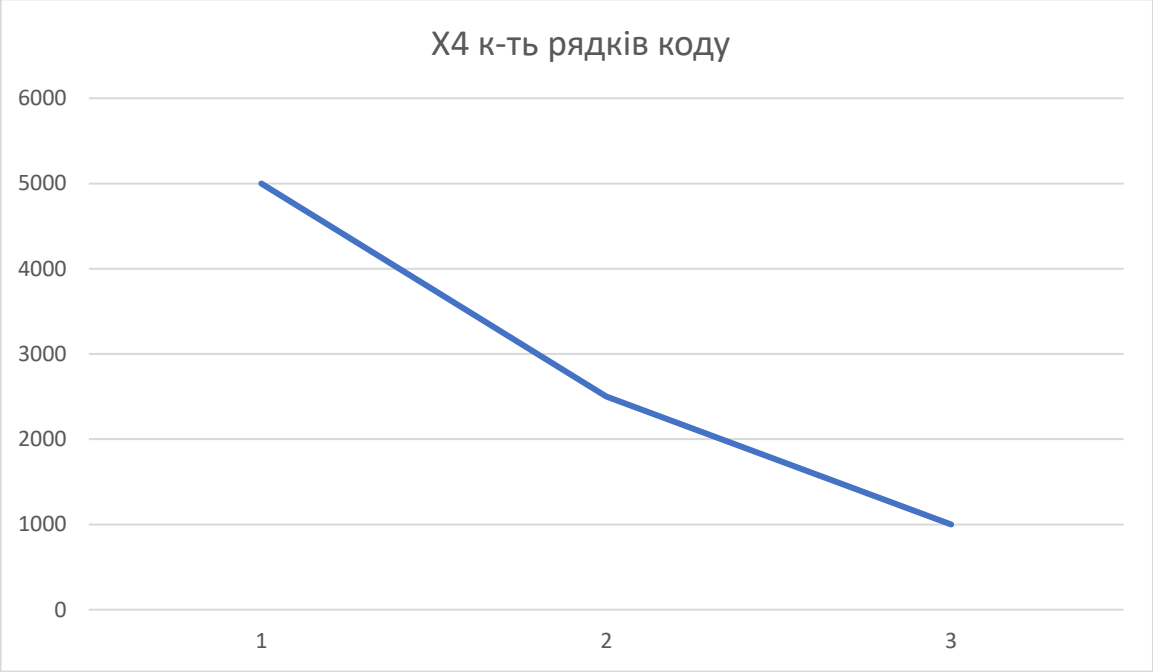

Рисунок 4.5 – X4, потенційний об’єм програмного коду

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.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. 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	Відхи-лення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	1	2	4	3	2	1	2	15	-6	36
X2	Об’єм пам’яті	Mb	4	5	3	3	6	3	6	30	9	81
X3	Час попередньої обробки даних	мс	2	3	2	2	1	3	1	14	-7	49
X4	Потенційний об’єм програмного коду	Кількість рядків коду	5	2	3	4	3	5	3	25	4	16
	Разом		12	12	12	12	12	12	12	84	0	182

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

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

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 84, \qquad (4.1)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 21 \tag{4.2}$$

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

$$\Delta_i = R_i - T. \tag{4.3}$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 182. \tag{4.4}$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 182}{7^2(4^3 - 4)} = 0{,}742 > W_k = 0{,}67. \tag{4.5}$$

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 і X2	<	<	>	=	<	<	<	<	0,5
X1 і X3	<	<	>	>	>	<	>	>	1,5
X1 і X4	<	=	>	<	<	<	<	<	0,5
X2 і X3	>	>	>	>	>	=	>	>	1,5

X2 і X4	<	>	=	<	>	<	>	<	0,5
X3 і X4	<	>	<	<	<	<	<	<	0,5

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

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \quad (4.6)$$

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

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

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \quad (4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \quad (4.8)$$

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:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \quad (4.9)$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \quad (4.10)$$

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 ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1	0,5	1,5	0,5	3,5	0,21	12,25	0,21	45,875	0,21
X2	1,5	1	1,5	0,5	4,5	0,28	16,25	0,28	59,125	0,28
X3	0,5	0,5	1	0,5	2,5	0,15	9,25	0,16	34,125	0,15
X4	1,5	1,5	1,5	1	5,5	0,36	21,25	0,35	77,875	0,36
Всього:					16	1	59	1	217	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Об'єм пам'яті), *X3* (час попередньої обробки даних) та *X4* (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \qquad (4.11)$$

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

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	12000	20	0,21	4,2
F3	А	X2	64	17	0,28	4,76
	Б	X3	128	16	0,15	2,4
F4	А	X4	1000	13	0,36	4,68

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \quad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 4{,}2 + 4{,}76 + 4{,}68 = 13{,}64;$$

$$K_{K2} = 4 + 3{,}8 + 8{,}28 = 11{,}28.$$

Як видно з розрахунків, кращим є 1 варіант, 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:

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.13)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

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

$K_М$ – коефіцієнт рівня мови програмування;

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

$K_{СТ.М}$ – 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_P =31 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.6. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ}$ = 0.8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 31 \cdot 1.6 \cdot 0.8 = 39{,}68 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_Р$ =25 людино-днів, $К_П$ =1.2, $К_{СК}$ = 1, $К_{СТ}$ =0.9:

$$T_2 = 25 \cdot \ 1.2 \cdot 0.9 = 27 \text{ людино-днів.}$$

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

$$T_I = (39{,}68 \ + 27 + 5.4 + 27) \cdot 8 = 792{,}64 \ \text{ людино-годин.}$$

$$T_{II} = (39{,}68 \ + 27 + 7.22 + 27) \cdot 8 = 807{,}2 \ \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 29526 грн., один аналітик в області даних з окладом 36908. Визначимо середню зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} \text{ грн.,} \qquad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_Ч = \frac{29526 \ + 29526 \ + 36908}{3 \cdot 21 \cdot 8} = 190{,}39 \ \text{ грн.} \qquad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_Ч \cdot T_i \cdot К_Д \, , \qquad (4.16)$$

де $С_Ч$– величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 190,39 \cdot 792,64 \cdot 1.2 = 181092,87$ грн.

II. $С_{ЗП} = 190,39 \cdot 807,2 \cdot 1.2 = 184419,36$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 181092,87 \cdot 0.22 = 39840,43$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 184419,36 \cdot 0.22 = 40572,25$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 29526 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 29526 \cdot 0,2 = 70862,4$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 70862,4 \cdot (1 + 0.2) = 85034,88$ грн.

Відрахування на соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.22 = 85034,88 \cdot 0,22 = 18707,67$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 27000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.2 \cdot 0.25 \cdot 27000 = 8100$ 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:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.2 \cdot 27000 \cdot 0.05 = 1620 \quad \text{грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 11) \cdot 8 \cdot 0.7 =$$

$$= 1355,2 \text{ години},$$

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:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1355,2 \cdot 0,5 \cdot 0,2 \cdot 4,86 = 658,62 \text{ грн.},$$

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:

$$С_Н = Ц_{ПР} \cdot 0.67 = 27000 \cdot 0,67 = 18090 \quad \text{грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \qquad (4.17)$$

$С_{ЕКС}$ = 85034,88 + 18707,67 + 8100 + 1620 + 658,62+ 18090 = 132211,17 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 132211,17 / 1355,2 = 97,55 \quad \text{грн/год.}$$

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:

$$С_М = С_{М-Г} \cdot Т, \quad (4.18)$$

I. $С_М = 97,55 \cdot 792,64 = 77322,03$ грн.

II. $С_М = 97,55 \cdot 807,2 = 78742,36$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67, \quad (4.19)$$

I. $С_Н = 181092,87 \cdot 0,67 = 121332,2$ грн.

II. $С_Н = 184419,36 \cdot 0,67 = 123560,97$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \quad (4.20)$$

I. $С_{ПП} = 181092,87 + 39840,43 + 77322,03 + 121332,2 = 419587,53$ грн.

II. $С_{ПП} = 184419,36 + 40572,25 + 78742,36 + 123560,97 = 427294,94$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР_j} = К_{Кj} / С_{Фj}, \quad (4.21)$$

$К_{ТЕР1} = 13,64 / 419587,53 = 3,2508 \cdot 10^{-5}$,

$К_{ТЕР2} = 11,28 / 427294,94 = 2,6398 \cdot 10^{-5}$.

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 3,2508 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, 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 $К_{ТЕР} = 3,2508 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

– Мова програмування – Python;
– Широкий вибір моделей, розширені можливості управління пам'яттю
– Інтерактивність, можливість візуалізації даних та результатів, зручність

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку та зручну реалізацію програми, доступний функціонал для роботи.

4.8 Висновки до четвертого розділу

В даній частині було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

У даній роботі проведено дослідження методів класифікації лейкоцитів на зображеннях крові мід мікроскопом з використанням згорткових нейронних мереж.

У першому розділі розглянуто актуальність задачі та перспективи розвитку. Описано та проаналізовано вхідні дані, а також наведено приклади зображень, що використовуються у дослідженні. Сформульовано задачу класифікації зображень, і надано відповідні твердження. Також сформульованао постановку задачі, а в кінці розділу описано технічні вимоги до системи.

У другому розділі приділено увагу математичним основам роботи. Наведені основні поняття згорткових нейронних мереж. Пояснено як комп'ютер "бачить" вхідні дані зображень. Наведено приклади базових шарів згорткових мереж, описані відомі оптимізатори та представлено методи попередньої обробки вхідних даних, що використовуються для поліпшення роботи мережі. Розглянуті основні метрики, які використовуються для задач класифікації зображень. Показані та описані існуючі архітектури згорткових нейронних мереж. Також створено власну архітектуру.

У третьому розділі зосереджено увагу на практичній частині роботи. Детально розглянуто вимоги до системи, на основі яких було обрано мови програмування та фреймворки для кожного компоненту системи. Описано процес попередньої обробки зображень, після чого проведено порівняльний аналіз навчених моделей, що в свою чергу дало змогу обрати найкращу модель, яку в подальшому було використано для роботи веб-додатку. У кінці було описано та продемонстровано роботу створеної системи.

У четвертому розділі проведено повний функціонально-вартісний аналіз програмного продукту, а також знайдено оцінку основних функцій програмного продукту. У результаті визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують. Обрано варіант реалізації програмного продукту.

СПИСОК ДЖЕРЕЛ

1. *White Blood Cell Definition & Function* (5 квітня, 2023) *Britannica*. Доступно за посиланням: https://www.britannica.com/science/white-blood-cell
2. *Lymphocytes* (дата відсутня) *Physiopedia*. Доступно за посиланням: https://www.physio-pedia.com/Lymphocytes
3. *Neutrophil* (6 травня, 2023) *The Editors of Encyclopedia of Britannica*. Доступно за посиланням: https://www.britannica.com/science/neutrophil
4. *Eosinophil* (25 лютого, 2023) *The Editors of Encyclopedia of Britannica*. Доступно за посиланням: https://www.britannica.com/science/eosinophil
5. *Monocytes* (28 Квітня, 2022) *Valerie Espinoza, Prabhu Emmady*. Доступно за посиланням: https://www.ncbi.nlm.nih.gov/books/NBK557618/
6. *Blood Cell Images Dataset* (2018) *Paul Mooney on Kaggle*. Доступно за посиланням: https://www.kaggle.com/datasets/paultimothymooney/blood-cells?sortBy=relevance&group=everyone&search=includeamin&pageSize=20&datasetId=9232
7. *What is the Difference Between Test and Validation Datasets?* (14 липня, 2017) *Jason Brownlee, Machine Learning Process.* Доступно за посиланням: https://machinelearningmastery.com/difference-test-validation-datasets/
8. *Convolutional Neural Networks, Explained* (26 Серпня, 2020) *Mayank Mishra in Towards Data Science.* Доступно за посиланням: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939
9. *Activation Functions in Neural Networks* (6 Вересня, 2017) *Sagar Sharma in Towards Data Science.* Доступно за посиланням: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

10. *What is Pooling in a Convolutional Neural Network (CNN): Pooling Layers Explained* (5 Грудня, 2021) *Seb.* Доступно за посиланням: https://programmathically.com/what-is-pooling-in-a-convolutional-neural-network-cnn-pooling-layers-explained/
11. *The Most Intuitive and Easiest Guide for Convolutional Neural Network* (24 Січня, 2019) *Jiwon Jeong.* Доступно за посиланням: https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480
12. *Fully Connected Layer: The brute force layer of a Machine Learning model* (дата відсутня) *Surya Pratap Singh.* Доступно за посиланням: https://iq.opengenus.org/fully-connected-layer/
13. *Gradient Descent in Machine Learning: A Basic Introduction* (27 Березня, 2023) *Niklas Donges.* Доступно за посиланням: https://builtin.com/data-science/gradient-descent
14. *A Comprehensive Guide on Optimizers in Deep Learning* (26 Квітня, 2023) *Ayush Gupta.* Доступно за посиланням: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/
15. *RProp* (8 Квітня, 2018) *Відсутня інформація про автора.* Доступно за посиланням: https://florian.github.io/rprop/
16. *Getting Started with Image Preprocessing in Python* (31 Серпня, 2021) *Adhinga Fredrick.* Доступно за посиланням: https://www.section.io/engineering-education/image-preprocessing-in-python/
17. *Image Preprocessing* (11 Жовтян, 2018) *Prince Canuma.* Доступно за посиланням: https://prince-canuma.medium.com/image-pre-processing-c1aec0be3edf
18. *What is the purpose of image preprocessing in deep learning?* (дата відсутня) *Відсутня інформація про автора.* Доступно за посиланням:

https://www.isahit.com/blog/what-is-the-purpose-of-image-preprocessing-in-deep-learning

19. *Image Processing using CNN: A beginners guide* (21 Червня, 2021) *Mohit Tripathi*. Доступно за посиланням: https://www.analyticsvidhya.com/blog/2021/06/image-processing-using-cnn-a-beginners-guide/
20. *Metrics to Use to Evaluate Deep Learning Object Detectors* (Серпень, 2020) *Venkatesh Wadawadagi*. Доступно за посиланням: https://www.kdnuggets.com/2020/08/metrics-evaluate-deep-learning-object-detectors.html
21. *Popular CNN Architectures Clearly Explained and Visualized* (22 Серпень, 2022) *Albers Uzila*. Доступно за посиланням: https://towardsdatascience.com/5-most-well-known-cnn-architectures-visualized-af76f1f0065e#8f20
22. *Deep Residual Networks (ResNet, ResNet50) – 2023 Guide* (2023) *Gaudenz Boesch*. Доступно за посиланням: https://viso.ai/deep-learning/resnet-residual-neural-network/
23. *Inception V1 Architecture Explained* (7 Жовтня, 2020) *Abheer Bandodker*. Доступно за посиланням: https://medium.com/@abheerchrome/inception-v1-architecture-explained-454b2eb66baf
24. *Everything you need to know about VGG16* (23 Вересня, 2021) *Great Learning*. Доступно за посиланням: https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

Файл із навчанням моделей – trainModel.py

```
import numpy as np
import pandas as pd
from scipy.spatial import distance as dist
import matplotlib.pyplot as plt
import os
import cv2
import seaborn as sns
from tqdm import tqdm
from sklearn.utils import shuffle
from sklearn import decomposition
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import tensorflow as tf
import keras
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from keras.models import Sequential, Model
from keras.applications import DenseNet201
from keras.initializers import he_normal
from keras.layers import Lambda, SeparableConv2D, BatchNormalization, Dropout,
MaxPooling2D, Input, Dense, Conv2D, Activation, Flatten
from keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
import imutils

def findEdges(image):
    # find edges in image
    gray = cv2.GaussianBlur(image, (1, 1), 0)
    edged = cv2.Canny(gray, 100, 400)
    edged = cv2.dilate(edged, None, iterations=1)
    edged = cv2.erode(edged, None, iterations=1)
    return edged

def getImgContours(edged):
    # find contours in the edge map
    contours = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    contours = imutils.grab_contours(contours)
    contours = sorted(contours, key=lambda x: cv2.contourArea(x))
    return contours

def getBoxes(contours, orig):
    # get the boxes
    boxes = []
```

```
    centers = []
    for contour in contours:
        box = cv2.minAreaRect(contour)
        box = cv2.cv.BoxPoints(box) if imutils.is_cv2() else cv2.boxPoints(box)
        box = np.array(box, dtype="int")
        (tl, tr, br, bl) = box
        if (dist.euclidean(tl, bl)) > 0 and (dist.euclidean(tl, tr)) > 0:
            boxes.append(box)
    return boxes

class_names = ['EOSINOPHIL', 'LYMPHOCYTE', 'MONOCYTE', 'NEUTROPHIL']
nb_classes = len(class_names)
image_size = (120,120)

def load_data():

    datasets = ['/kaggle/input/blood-cells/dataset2-master/dataset2-
master/images/TRAIN','/kaggle/input/blood-cells/dataset2-master/dataset2-
master/images/TEST' ]
    images = []
    labels = []

    count =0
    for dataset in datasets:

        for folder in os.listdir(dataset):

            if folder in ['EOSINOPHIL']: label = 0
            elif folder in ['LYMPHOCYTE']: label = 1
            elif folder in ['MONOCYTE']: label = 2
            elif folder in ['NEUTROPHIL']: label = 3

            for file in tqdm(os.listdir(os.path.join(dataset, folder))):

                img_path = os.path.join(os.path.join(dataset, folder), file)

                image = cv2.imread(img_path)
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

                image =
cv2.copyMakeBorder(image,10,10,10,10,cv2.BORDER_CONSTANT,value=[198, 203, 208])

                image1 = cv2.inRange(image,(80, 80, 180),(180, 170, 245))

                kernel = np.ones((3, 3), np.uint8)
                kernel1 = np.ones((5, 5), np.uint8)
                img_erosion = cv2.erode(image1, kernel, iterations=2)
                image1 = cv2.dilate(img_erosion, kernel1, iterations=5)

                edgedImage = findEdges(image1)
```

```
                edgedContours = getImgContours(edgedImage)
                edgedBoxes =  getBoxes(edgedContours, image.copy())
                if len(edgedBoxes)==0:
                    count +=1
                    continue

                last = edgedBoxes[-1]
                max_x = int(max(last[:,0]))
                min_x = int( min(last[:,0]))
                max_y = int(max(last[:,1]))
                min_y = int(min(last[:,1]))

                mask = np.zeros_like(image)
                cv2.drawContours(mask, edgedContours, len(edgedContours)-1,
(255,255,255), -1)

                image[mask==0] = 0

                image = image[min_y:max_y, min_x:max_x]

                if (np.size(image)==0):
                    count +=1
                    continue
                image = cv2.resize(image, image_size)

                images.append(image)
                labels.append(label)

    images = np.array(images, dtype = 'float32')
    labels = np.array(labels, dtype = 'int32')

    return images, labels

images, labels = load_data()
images, labels = shuffle(images, labels, random_state=10)
train_images, test_images, train_labels, test_labels = train_test_split(images,
labels, test_size = 0.2)
test_images, val_images, test_labels, val_labels = train_test_split(test_images,
test_labels, test_size = 0.5)

base_model1 = tf.keras.applications.resnet50.ResNet50(
  include_top=False,
  weights='imagenet'
)
base_model1.trainable = False
base_model2 = tf.keras.applications.inception_v1.InceptionV1(
  include_top=False,
  weights='imagenet'
)
base_model2.trainable = False
```

```
base_model3 = tf.keras.applications.vgg16.VGG16(
  include_top=False,
  weights='imagenet'
)
base_model3.trainable = False


model1 = Sequential()

model1.add(Conv2D(16 , (3,3) , padding = 'same' , activation = 'relu' , input_shape
= (120,120,3)))
model1.add(Conv2D(16 , (3,3), padding = 'same' , activation = 'relu'))
model1.add(MaxPooling2D(pool_size = (2,2)))

model1.add(SeparableConv2D(32, (3,3), activation = 'relu', padding = 'same'))
model1.add(SeparableConv2D(32, (3,3), activation = 'relu', padding = 'same'))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size = (2,2)))

model1.add(SeparableConv2D(64, (3,3), activation = 'relu', padding = 'same'))
model1.add(SeparableConv2D(64, (3,3), activation = 'relu', padding = 'same'))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size = (2,2)))

model1.add(SeparableConv2D(128, (3,3), activation = 'relu', padding = 'same'))
model1.add(SeparableConv2D(128, (3,3), activation = 'relu', padding = 'same'))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size = (2,2)))
model1.add(Dropout(0.2))

model1.add(SeparableConv2D(256, (3,3), activation = 'relu', padding = 'same'))
model1.add(SeparableConv2D(256, (3,3), activation = 'relu', padding = 'same'))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size = (2,2)))
model1.add(Dropout(0.2))

model1.add(Flatten())
model1.add(Dense(units = 512 , activation = 'tanh'))
model1.add(Dropout(0.7))
model1.add(Dense(units = 128 , activation = 'tanh'))
model1.add(Dropout(0.5))
model1.add(Dense(units = 64 , activation = 'tanh'))
model1.add(Dropout(0.3))

model1.add(Dense(units = 4 , activation = 'softmax'))

model1.compile(optimizer = "Adam", loss = 'sparse_categorical_crossentropy' ,
metrics = ['accuracy'])
model1.summary()
```

```
checkpoint = ModelCheckpoint(filepath='best_model.hdf5', save_best_only=True,
save_weights_only=False)
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.1, patience=3, verbose =
1, mode='min', restore_best_weights = True)
learning_rate_reduction = ReduceLROnPlateau(
    monitor = 'val_accuracy',
    patience = 2,
    verbose = 1,
    factor = 0.3,
    min_lr = 0.000001)

history1 = model1.fit(
    train_images,
    train_labels,
    batch_size = 32,
    epochs = 30,
    validation_data=(val_images, val_labels),
    callbacks=[learning_rate_reduction])

model1.save('/kaggle/working/leukocyte_classification_model.h5')
```

Файл backend частини – main.py

```
from fastapi import FastAPI, File, UploadFile
from fastapi.responses import JSONResponse
import cv2
import numpy as np
from scipy.spatial import distance as dist
import tensorflow as tf
from PIL import Image
import io
from fastapi.middleware.cors import CORSMiddleware
import sys
import imutils

model_path = "leukocyte_classification_model.h5"
model = tf.keras.models.load_model(model_path)

app = FastAPI()

def find_edges(image):
    gray = cv2.GaussianBlur(image, (1, 1), 0)
    edged = cv2.Canny(gray, 100, 400)
    edged = cv2.dilate(edged, None, iterations=1)
    edged = cv2.erode(edged, None, iterations=1)
    return edged

def get_img_contours(edged):
```

```
    contours = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    contours = contours[0] if len(contours) == 2 else contours[1]
    contours = sorted(contours, key=lambda x: cv2.contourArea(x))
    return contours

def get_boxes(contours, orig):
    boxes = []
    for contour in contours:
        box = cv2.minAreaRect(contour)
        box = cv2.boxPoints(box) if cv2.__version__.startswith('3') else
cv2.boxPoints(box)
        box = np.array(box, dtype="int")
        (tl, tr, br, bl) = box
        if (dist.euclidean(tl, bl)) > 0 and (dist.euclidean(tl, tr)) > 0:
            boxes.append(box)
    return boxes

@app.post("/upload-image")
async def upload_image(file: UploadFile = File(...)):
    contents = await file.read()
    nparr = np.frombuffer(contents, np.uint8)
    img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    return JSONResponse(status_code=200, content={"message": "Image uploaded
successfully"})

@app.post("/classify")
async def classify_image(image: UploadFile = File(...)):
    contents = await image.read()
    nparr = np.frombuffer(contents, np.uint8)
    img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    edged_image = find_edges(img)
    edged_contours = get_img_contours(edged_image)
    edged_boxes = get_boxes(edged_contours, img.copy())

    last = edged_boxes[-1]
    max_x = int(max(last[:, 0]))
    min_x = int(min(last[:, 0]))
    max_y = int(max(last[:, 1]))
    min_y = int(min(last[:, 1]))

    prediction = model.predict(image)
    class_index = np.argmax(prediction)
    class_label = str(class_index)
```

```
    return JSONResponse(status_code=200, content={"x1": min_x, "y1": min_y, "x2":
max_x, "y2": max_y, "class_label" : class_label})
```

Файл frontend частини – App.js

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

const App = () => {
  const [selectedImage, setSelectedImage] = useState(null);
  const [classifyClicked, setClassifyClicked] = useState(false);
  const [classificationData, setClassificationData] = useState(null);

  const handleImageUpload = (event) => {
    const file = event.target.files[0];
    setSelectedImage(file);
  };

  const handleClassifyClick = async () => {
    setClassifyClicked(true);
    const response = await fetch('http://localhost:8000/classify', {
      method: 'POST',
      body: selectedImage,
    });
    const data = await response.json();
    setClassificationData(data);
  };

  return (
    <div style={{ fontFamily: 'Arial', color: 'darkblue', textAlign: 'center',
padding: '20px' }}>
      {!selectedImage && !classifyClicked && (
        <div>
          <h2>Please select an image:</h2>
        </div>
      )}

      {!classifyClicked && (
        <div>
          <input type="file" id="image-upload" style={{ display: 'none' }}
onChange={handleImageUpload} />
          <label htmlFor="image-upload" style={{ cursor: 'pointer', padding: '10px
20px', background: 'lightblue', borderRadius: '5px', margin: '10px' }}>
            Select Image
          </label>
        </div>
      )}

      {selectedImage && (
        <div>
```

```
          <h2>Selected image:</h2>
          <div style={{ marginBottom: '20px' }}>
            <img src={URL.createObjectURL(selectedImage)} alt="Selected" style={{
maxWidth: '100%', maxHeight: '400px' }} />
          </div>
        </div>
      )}
      {classifyClicked && (
        <div>
          <h2>Classification Result:</h2>
          <div style={{ marginBottom: '20px', position: 'relative' }}>
            {selectedImage && (
              <img src={URL.createObjectURL(selectedImage)} alt="Uploaded" style={{
maxWidth: '100%', maxHeight: '400px' }} />
            )}
            {classificationData && (
              <div
                style={{
                  position: 'absolute',
                  top: classificationData.y_min,
                  left: classificationData.x_min,
                  width: classificationData.x_max - classificationData.x_min,
                  height: classificationData.y_max - classificationData.y_min,
                  border: '2px solid red',
                }}
              />
            )}
          </div>
          <p style={{ fontSize: '20px', fontWeight: 'bold' }}>{classificationData
&& classificationData.class_label}</p>
        </div>
      )}

      {selectedImage && !classifyClicked && (
        <div>
          <label htmlFor="classify-btn" style={{ cursor: 'pointer', padding: '10px
20px', background: 'lightblue', borderRadius: '5px', margin: '10px' }}>
            <button id="classify-btn" style={{ display: 'none' }}
onClick={handleClassifyClick}>
              Classify
            </button>
            Select Image
          </label>
        </div>
      )}
    </div>
  );
};

export default App;
```