МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____Юрій КОНДРАТЕНКО

«____»_____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

ІНТЕЛЕКТУАЛЬНА СИСТЕМА ІДЕНТИФІКАЦІЇ ТА КЛАСИФІКАЦІЇ ВІЙСЬКОВОЇ ТЕХНІКИ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Інтелектуальні інформаційні системи»

Здобувач _____ Олександр ГЕРАС

«___»_____ 2024 р.

Керівник канд. техн. наук, доцент _____ Євген СІДЕНКО

«___»_____ 2024 р.

Миколаїв – 2024

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Другий (магістерський)
Освітній ступень	Магістр
Спеціальність	122 Комп'ютерні науки
Освітня програма	Інтелектуальні інформаційні системи

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Юрій КОНДРАТЕНКО

«____» _____ 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

_____Гераса Олександра Миколайовича_____
((прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Інтелектуальна система ідентифікації та класифікації військової техніки з використанням нейромереж».

Керівник роботи: Сіденко Євген Вікторович, доцент кафедри інтелектуальних інформаційних систем, канд. техн. наук, доцент.

Затверджена наказом ЧНУ ім. Петра Могили від «03» червня 2024 р. № 140/1.

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

3. Вхідні (початкові) дані до роботи: сфера нейромереж для завдань з класифікації військової техніки на фотографіях, набір даних з фотографіями військової техніки різних типів.

Очікуваний результат роботи: вебзастосунок для класифікації військової техніки на фотографіях.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

– аналіз предметної сфери нейронних мереж та дослідження теоретичних засад класифікації зображень;

– обгрунтування вибору технологій і засобів розробки системи класифікації об’єктів на фотографіях;

– проектування та розробка програмної реалізації вебзастосунку для класифікації об’єктів на фотографіях, дослідження якості моделі класифікації об’єктів та точності класифікації.

5. Перелік графічних матеріалів: презентація.

Керівник роботи _____ Євген СІДЕНКО
(Особистий підпис) *(Власне ім’я ПРІЗВИЩЕ)*

Здобувач _____ Олександр ГЕРАС
(Особистий підпис) *(Власне ім’я ПРІЗВИЩЕ)*

Дата видачі завдання «07» червня 2024 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Інтелектуальна система ідентифікації та класифікації військової техніки з використанням нейромереж

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	03.06.2024	07.06.2024	Виконано
2	Аналіз предметної області та постановка задачі	10.06.2024	20.06.2024	Виконано
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема аналіз існуючих публікацій на тему класифікації військової техніки за допомогою нейромереж	21.06.2024	01.07.2024	Виконано
4	Огляд архітектур штучних нейронних мереж для вирішення поставленої задачі	01.09.2024	25.10.2024	Виконано
5	Створення моделей обраних нейромереж	26.10.2024	21.11.2024	Виконано
6	Перший попередній захист КР на засіданні комісії кафедри	22.11.2024	22.11.2024	Виконано
7	Корегування роботи за результатами попереднього захисту	23.11.2024	05.12.2024	Виконано
8	Другий попередній захист КР на засіданні комісії кафедри	06.12.2024	06.12.2024	Виконано
9	Доробка та остаточне оформлення КР	07.12.2024	10.12.2024	Виконано
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	16.12.2024	17.12.2024	Виконано

Керівник роботи _____ Євген СІДЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Олександр ГЕРАС
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«19» червня 2024 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 601м ЧНУ ім. Петра Могили

Гераса Олександра Миколайовича

на тему: **“ІНТЕЛЕКТУАЛЬНА СИСТЕМА ІДЕНТИФІКАЦІЇ ТА КЛАСИФІКАЦІЇ ВІЙСЬКОВОЇ ТЕХНІКИ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ”**

Актуальність данного дослідження полягає в важливості цього завданням у сфері комп'ютерного зору та у сучасному світі. Використання глибоких нейронних мереж для класифікації різних типів техніки набуло широкого поширення, оскільки такі моделі дозволяють обробляти складні структури зображень та досягати високої точності. Особливості сучасних нейромереж, зокрема здатність тренувати глибокі моделі з великою кількістю шарів, роблять їх ефективним інструментом для задач, де важлива точна ідентифікація та класифікація об’єктів. Результати таких досліджень можуть бути застосовані в оборонній сфері та інших галузях, де обробка та класифікація зображень є критичним завданням.

Об’єктом дослідження є процес класифікації військової техніки за допомогою нейронних мереж.

Предметом дослідження є нейронні мережі для класифікації військової техніки на фотографіях.

Метою дослідження є підвищення точності класифікації об'єктів на фотографіях за рахунок застосування моделей нейронних мереж. Основними цілями дослідження є вивчення принципів функціонування різних видів нейромереж, аналіз їх особливостей порівняно з іншими архітектурами глибоких нейронних мереж, оцінка їх ефективності та точності у класифікації військової техніки на фотографіях.

Робота буде включати експериментальну частину з використанням відповідних наборів даних для тренування та тестування нейромереж.

Отримані результати дослідження сприятимуть розумінню можливостей та обмежень обраних нейромереж у контексті класифікації зображень, що може мати практичне значення у галузі комп'ютерного зору та штучного інтелекту.

Робота складається з фахового розділу і спеціальної частини з охорони праці. Пояснювальна записка складається зі вступу, чотирьох розділів, висновків та додатків.

У першому розділі розглядаються технології для класифікації об’єктів на фото, аналіз аналогічних систем та формування завдання на КР.

У другому розділі досліджено основні принципи роботи нейронних мереж, що використовуватимуться в дослідженні, їх архітектура та алгоритми навчання.

У третьому розділі описано процес формування даних для навчання нейромереж, їх обробка та сортування. Також було спроектовано процес по створення вебзастосунку для роботи з класифікацією фото.

У четвертому розділі наведено результати ствренння нейромереж, створення вебзастосунку та їх тестування.

В результаті розроблено систему класифікації військової техніки на фотографіях з використанням нейромережевих техноло гій з точністю близько 95%.

Кваліфікаційна робота містить 87 сторінок, 33 рисунок та 25 використаних джерел.

Ключові слова: нейронна мережа, ResNet, DenseNet, класифікація об'єктів, фотографії, глибоке навчання, точність класифікації, нейрон, ядро, шар, архітектура.

ABSTRACT

to the qualification work by the student of the group 601m of Petro Mohyla Black Sea National University

Geras Oleksandr Mykolayovych

"INTELLIGENT SYSTEM OF IDENTIFICATION AND CLASSIFICATION OF MILITARY EQUIPMENT USING NEURAL NETWORKS"

The relevance of this study lies in the importance of this task in the field of computer vision and in the modern world. The use of deep neural networks for classifying various types of machinery has become widespread, as such models allow processing complex image structures and achieving high accuracy. The features of modern neural networks, in particular the ability to train deep models with a large number of layers, make them an effective tool for tasks where accurate identification and classification of objects is important. The results of such research can be applied in the defense sector and other industries where image processing and classification is a critical task.

The object of research is the process of classifying military equipment using neural networks.

The subject of the study is neural networks for classifying military equipment in photographs.

The purpose of the study is to improve the accuracy and classification of objects in photographs by using neural network models. The main objectives of the study are to study the principles of functioning of different types of neural networks, analyze their features in comparison with other deep neural network architectures, and evaluate their efficiency and accuracy in classifying military equipment in photographs.

The work will include an experimental part using appropriate datasets for training and testing neural networks.

The obtained results of the study will contribute to understanding the capabilities and limitations of the selected neural networks in the context of image classification, which may be of practical importance in the field of computer vision and artificial intelligence.

The work consists of a professional section and a special part on labor protection. The explanatory note consists of an introduction, four chapters, conclusions, and appendices.

The first section discusses technologies for classifying objects in photos, analyzing similar systems, and forming a task for research.

The second section describes the basic principles of neural networks to be used in the BCR, their architecture and training algorithms.

The third section describes the process of generating data for neural network training, processing, and sorting. We also designed the process of creating a web application for photo classification.

Section 4 presents the results of neural network training, web application development, and testing.

As a result, a system for classifying military vehicles in photographs using neural network technologies with an accuracy of about 95% was developed.

Master's thesis consists of 87 pages, 33 figure and 25 references.

Keywords: neural network, ResNet, DenseNet, object classification, photos, deep learning, classification accuracy, neuron, kernel, layer, architecture.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ

ШІ — штучний інтелект

CNN — Convolutional neural network

ResNet — Residual Neural Network

RNN — Recurrent neural network

CNTK — Microsoft Cognitive Toolkit

API — Application Programming Interface

GCN — Graph Neural Network

SMRN — Semantic Modulation Residual Network

NTM — Neural Turing Machine

SGD — Stochastic Gradient Descent

DNN — deep neural network

VGG — Visual Geometry Group

JSON — JavaScript Object Notation

ВСТУП

У сучасному світі, де обсяги інформації зростають щосекунди, використання штучного інтелекту для автоматизованої обробки даних набуває ключового значення. Нейронні мережі сьогодні є потужними інструментами впливу на людей та засобами для вирішення численних завдань, зокрема класифікації об’єктів на зображеннях.

Одним із найефективніших підходів до аналізу зображень стало використання глибоких нейронних мереж та рекурентних нейронних мереж, що здатні виявляти складні зв'язки в даних і забезпечувати високоточну класифікацію.

Ця кваліфікаційна робота має на меті підвищити точність класифікації військової техніки на фото через використання моделей нейронних мереж. Дослідження включає аналіз методів та технологій для побудови й тренування глибоких нейронних мереж, детальний огляд архітектур ResNet та DenseNet і на кінець влючено практичне закріплення всього вивченого матеріалу з використанням даних із мережі Інтеренет.

Результатом дослідження планується отримати дані, що статуть в пригоді для галузі нейронних мереж, а головне в військовій сфері діяльності, або ж для підвищення точності класифікації зображень у таких сферах, як медицина, транспорт, маркетинг та бізнес-аналітика.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ФОРМУЛЮВАННЯ ЗАВДАННЯ

1.1 Головні визначення та поняття

У цій роботі все дослідження зосереджене на комп'ютерному зорі – технології, яка надає комп'ютерам здатність "бачити" і розуміти світ навколо нас так само, як це робимо ми. Це досягається за допомогою складних алгоритмів, які дозволяють комп'ютерам аналізувати зображення і відео, виявляючи об'єкти, розпізнаючи обличчя, відстежуючи рух та виконуючи багато інших завдань [2].

Комп'ютерний зір знаходить застосування в таких областях, як класифікація об’єктів різних видів на фото та відео матеріалах, відстежування рухаючихся об’єктів, ідентифікація людей по обличчю, а також у медицині та робототехніці. Одним з найцікавіших досягнень в методах застосування комп’ютерного зору є відновленння тривимірної сцени по одному чи декільком зображенням. Зі зростанням методі доступу до фото та відео матеріалів, швидкісним розширенням потужностей комп’ютерного устаткування та різного ПЗ, computer vision перетворюється на більш важливу та значущю сферу з великою кількість можливих методів застосування в дослідженнях, аеросфері і в житті багатьох жителів Землі [3].

Комп'ютерний зір — це, по суті, навчання на прикладах. Для того, щоб навчити систему розпізнавати, наприклад, автомобільні шини, їй необхідно проаналізувати велику кількість фото шин та інших схожих об'єктів. Повторюючи цей процес багато разів, система поступово вчиться виокремлювати характерні риси шин, дозволяючи їй точно ідентифікувати їх навіть за відсутності недоліків [3, 4].

Потім преведено кілька прикладів простих завдань, для computer vision:

– класифікація зображень аналізує зображення та може визначити його клас (наприклад, груша, метелик, кішка). Тобто, модель здатна точно передбачити,

до якого класу належить зображення. Соціальні мережі, наприклад, можуть використовувати класифікацію для автоматичного виявлення та фільтрації неякісних фото, наданих клієнтами;

– object detection використовує класифікацію об’єктів на фото для класифікації конкретного класу, а потім знаходить ці об'єкти на зображеннях чи у відео. Це застосовується, наприклад, для виявлення дефектів на виробничій лінії або для ідентифікації устаткування, що потребує обслуговування;

– відстеження об'єктів забезпечує спостереження за їхнім переміщенням після початкового виявлення, зазвичай у послідовних кадрах або під час потокової обробки відео. Наприклад, автономні транспортні засоби повинні не лише ідентифікувати та розпізнавати об'єкти, такі як пішоходи, автомобілі чи дорожні знаки, але й точно відстежувати їх рух для запобігання зіткненням і дотримання правил дорожнього руху;

– технологія пошуку зображень за змістом використовує алгоритми комп’ютерного зору для аналізу, пошуку й отримання зображень із великих баз даних, орієнтуючись на їхній вміст, а не на метадані. Цей процес може включати автоматичне створення анотацій для зображень, що замінює ручне тегування. Це покращує точність управління цифровими ресурсами та полегшує доступ до необхідної інформації.

Кількість зображень, потрібних для навчання нейронної мережі, залежить від таких факторів, як складність завдання, кількість категорій, обсяг елементів у кожній категорії та параметри навчання, зокрема кількість епох. Для забезпечення високої точності класифікації зазвичай рекомендується використовувати щонайменше 200 зображень на кожну категорію. Наприклад, для навчання моделі, яка класифікує три типи об'єктів, необхідно мінімум 600 зображень (по 200 для кожного типу). У цьому дослідженні для кожного типу техніки було використано саме по 200 зображень.

Головним матеріалом для створення будь-якої нейромережі є нейрон, що є без перебільшення головним будівельним матеріалом будь-якої штучної нейронної мережі. Він отримує дані на вхід, далі розраховує деякі дії з цими даними та в результаті цих маніпуляцій генерує вихідні дані. В склад нейрону входять деякі компоненти, включно з вхідними взаємозв’язками, головний його елемент - функцію активації ну і на останок вихідний сигнал.

Ваги — це числа, які визначають силу з'єднання між нейронами. Під час навчання мережі ваги змінюються, щоб оптимізувати її роботу. За активаційну функцію відповідає нелінійна функція, яка застосовується до суми зважених входів. Поширені активаційні функції: сигмоїда, ReLU, тангенс гіперболічний тощо. Але треба зауважити, що деякі нейрони мають поріг, який необхідно перевищити, щоб нейрон активувався.

Кожен нейрон приймає дані через свої вхідні зв'язки, кожен із яких характеризується певною вагою. Далі нейрон обчислює загальну зважену суму отриманих даних, після чого застосовує функцію активації, яка виконує нелінійну обробку цієї суми. Результат роботи функції стає вихідним сигналом нейрона, що передається іншим нейронам для подальших обчислень.

Штучні нейрони в нейронних мережах застосовуються для розв'язання різних задач, зокрема класифікації, розпізнавання образів, автоматичного перекладу та інших [7].

Одною з головних моделей використаних в дослідженні нейромереж є архітектура моделі згорткової нейронної мережі (CNN), котру запропонував Лекуном та ін., була спеціально розроблена для обробки зображень шляхом введення таких відмінних характеристик, як локальні рецептивні поля та спільні ваги, що допомагає зменшити перелік параметрів і обчислювальну складність. До відомих моделей CNN належать AlexNet, VGG, ResNet, DenseNet тощо. Ці мережі CNN та їх варіації широко застосовувалися для виявлення поліпів на відеозаписах

колоноскопії, виявлення інтерстиціальних вузликів у легенях, класифікації зображень клітин гепатиту-2, класифікації утворень у грудях, виявлення біомаркерів захворювань у дрібних судинах головного мозкута класифікації раку шкіри.

Хоча ШНМ загалом працює краще, ніж традиційні методи вилучення ознак, він вимагає великої кількості даних зображень для тривалого процесу навчання під наглядом, а також пов'язаний з труднощами у визначенні кількості ядер згортки і вразливістю до параметрів і манер навчання. Перші можуть збільшити кількість даних шляхом обертання, перекладу тощо, але багато згенерованих зображень все одно залишаються досить схожими, що впливає на ефективність методу доповнення даних. Навчання з перенесенням добре працює для мереж з великими вагами, такими як AlexNet і ResNet-50, але встановлення параметрів і точне налаштування все ще вимагає багато випробувань.

Одним із ключових об’єктів дослідження є ResNet (Residual Network) — архітектура глибокої нейронної мережі, представлена у 2015 році науковцями Microsoft Research. Ця згорткова нейронна мережа (CNN) вирізняється здатністю досягати значної глибини, успішно долаючи проблему зникнення градієнтів, яка зазвичай виникає під час навчання дуже глибоких моделей [6].

Методи глибокого навчання базуються на великих наборах маркованих даних і багатошарових нейронних мережах, що дозволяє ефективно вирішувати складні завдання. Вони демонструють високий рівень точності, але потребують значних обсягів даних для тренування.

Класифікація зображень є одним із практичних застосувань глибокого навчання, що передбачає автоматичне розподілення зображень за категоріями. Наприклад, визначення наявності обличчя на фото чи класифікація зображень за видами тварин (наприклад, розрізнення котів і собак).

Головним проривом в архітектурі ResNet є використання залишкових з’єднань, що дають змогу передавати інформацію з початкових шарів безпосередньо до виходу, оминаючи проміжні шари. Це усуває труднощі з передаванням градієнтів, значно спрощуючи навчання нейронних мереж великої глибини.

Він показав ефективну роботу виконуючи різні завданнях комп'ютерного зору, як класифікація об’єктів на фото, детекція об'єктів і сегментація. Більше того, він став основою для численних передових моделей, що значно підвищили їх продуктивність і встановили нові стандарти якості [5].

Було розроблено кілька варіацій архітектури ResNet з різною глибиною та структурою. Найбільш популярні моделі ResNet наведено далі.

ResNet-18 є компактною та легкою архітектурою в сімействі ResNet, яка включає всього 18 шарів. Вона базується на залишкових (residual) блоках, кожен із яких складається з 2-3 шарів згортки. Завдяки своїй невеликій глибині, ця модель має нижчі обчислювальні витрати та використовується для задач, де простота і швидкість виконання мають вирішальне значення. ResNet-18 добре підходить для застосунків із обмеженими ресурсами, таких як вбудовані системи чи мобільні пристрої.

ResNet-34 є розширеною версією ResNet-18, що включає 34 шари. Вона використовує більш глибокі залишкові блоки, які містять 3-4 шари згортки в кожному блоці. Завдяки цьому збільшується здатність моделі до розпізнавання складних особливостей у даних. ResNet-34 пропонує баланс між продуктивністю та ефективністю, що робить її популярною для задач середньої складності, таких як класифікація великих зображень чи розпізнавання об'єктів.

ResNet-50 є однією з найпопулярніших архітектур у сімействі ResNet завдяки своїй потужності та універсальності. Вона містить 50 шарів, структурованих у вигляді залишкових блоків, кожен із яких включає три згорткові

шари. Ця модель стала першою, яка досягла найсучасніших результатів у задачі класифікації ImageNet, демонструючи високу точність. ResNet-50 широко використовується для різноманітних задач комп'ютерного зору, включаючи класифікацію зображень, сегментацію, розпізнавання об'єктів і навіть у переносному навчанні для спеціалізованих застосунків.

ResNet-101 є глибшою версією ResNet-50, яка складається зі 101 шару. Подібно до ResNet-50, вона використовує залишкові блоки з трьома згортковими шарами в кожному блоці, але має більшу кількість таких блоків, що дозволяє моделі розпізнавати ще складніші патерни у даних. ResNet-101 забезпечує більш високу точність, ніж ResNet-50, і використовується для більш складних завдань комп'ютерного зору, таких як семантична сегментація чи детекція об'єктів у великих наборах даних.

ResNet-152 є найбільшою архітектурою в класичній серії ResNet, яка включає 152 шари. Вона також використовує залишкові блоки з трьома згортковими шарами в кожному, але завдяки значній глибині здатна виявляти дуже складні особливості у даних. Ця модель зазвичай застосовується для найскладніших задач комп'ютерного зору, де потрібна максимальна точність, наприклад, у медичній діагностиці, класифікації супутникових зображень чи аналізі відео. Однак через велику кількість параметрів і обчислювальну складність її використання вимагає значних апаратних ресурсів.

DenseNet (Dense Convolutional Network) — це архітектура глибокої нейронної мережі, запропонована для подолання деяких недоліків традиційних мереж, як-от ResNet. Основна ідея DenseNet полягає в щільному зв'язку між шарами мережі: кожен шар отримує вхід не лише від попереднього шару, а й від усіх попередніх шарів мережі. Така щільна архітектура не лише сприяє покращенню передачі градієнтів, але й робить мережу більш ефективною,

зменшуючи кількість параметрів, оскільки кожен шар має доступ до "колективного досвіду" усіх попередніх шарів.

Особливості архітектури DenseNet:

– блоки щільних зв’язків (Dense Blocks) — у кожному блоці кожен шар з’єднаний із кожним іншим шаром через прямі зв’язки. Це дозволяє моделі ефективніше використовувати інформацію та спрощує навчання.

– тонші шари завдяки щільним зв’язкам мережа потребує менше каналів для кожного шару, що значно зменшує загальну кількість параметрів.

– перехідні шари (Transition Layers) — їх розміщують між блоками, вони використовують згортку 1x1 і операцію пулінгу для зменшення розмірів, щоб зменшити розмірність і контролювати кількість характеристик.

DenseNet має переконливі переваги: вона заохочує повторне використання елементів і полегшує проблему зникаючого градієнта. Однак він також має очевидні недоліки. По-перше, кожен шар просто об'єднує карти особливостей, отримані з попередніх шарів за допомогою операції конкатенації, не враховуючи взаємозалежності між різними каналами. Ми вважаємо, що шляхом вдосконалення моделі, моделювання кореляції каналів ознак та реалізації рекалібрування ознак каналів, представлення мережі можна ще більше покращити. По-друге, кореляція міжшарової карти ознак не моделюється явно. Дуже корисно адаптивно вивчати коефіцієнти кореляції, моделюючи кореляцію карт ознак між шарами.

Архітектура DenseNet є більш ефективною з погляду використання ресурсів і забезпечує високу продуктивність у багатьох завданнях комп'ютерного зору.

Для мережі з L шарами традиційна згорткова мережа мала б L з'єднань, а ResNet — 2L з'єднань. Однак у DenseNet на кожен шар існують прямі з'єднання, що пов'язують його з усіма наступними шарами, що призводить до L(L+1)/2 з'єднань. Це можна побачити на рисунку 1.

У DenseNet схожі відображення з'єднуються через конкатенацію, щоб ефективніше зберігати інформацію.

Іншими словами, шар l одержує ознаки від попередніх шарів і об’єднує їх перед тим, як подати на нелінійне перетворення H□(·).

У початковій статті про DenseNet функція H□(·) складалася з трьох послідовних операцій: нормалізації пакету (BN), за якою слідує виправлена лінійна функція (ReLU), а потім згортка 3x3. Ці елементи складають те, що називають Щільним Блоком. Схематично це представлено на рисунку 3.

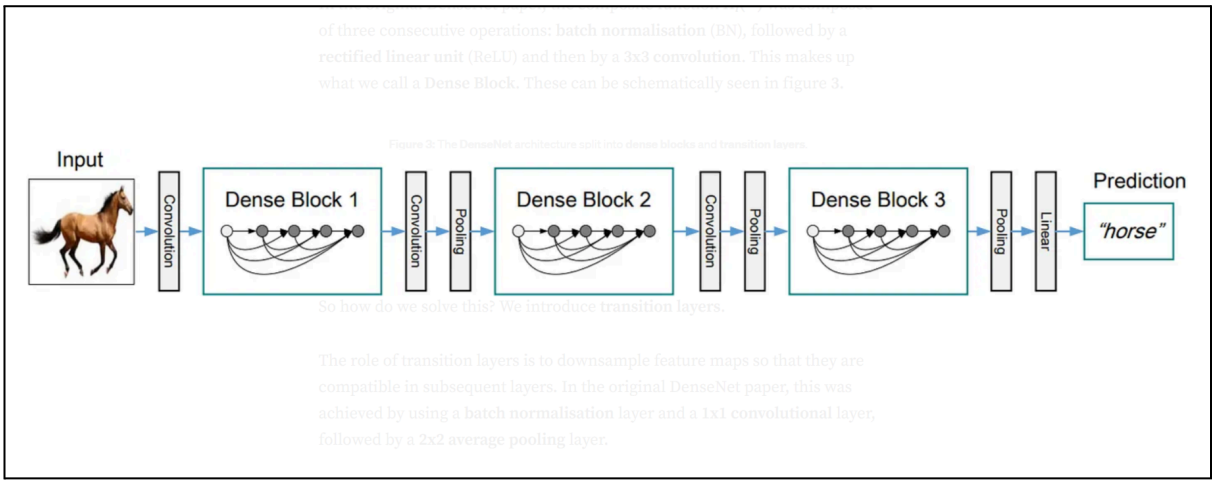

Рисунок 1.1 — Архітектура DenseNet

Архітектура DenseNet, поділена на щільні блоки та перехідні шари. Через природу згорток розмір карт ознак зменшується на кожному шарі. Це створює проблеми сумісності при конкатенації карт ознак з різних шарів.

Перехідні шари мають роль зменшення розмірів карт ознак, щоб вони були сумісні на наступних шарах. У початковій статті DenseNet це досягалося використанням шару нормалізації пакету та шару згортки 1x1, за якими слідує середнє пулінгування 2x2.

Хоча на перший погляд може здатися, що DenseNet потребує більше параметрів, ніж її попередники, насправді це не так. Це обумовлено швидкістю зростання DenseNet і шаром-вузьким місцем (bottleneck layer).

Keras — це популярна бібліотека для побудови нейронних мереж, створена на мові Python. Вона була розроблена Франсуа Шолле та представлена публіці у 2015 році. Завдяки своїй простоті й ефективності Keras став одним із головних інструментів для фахівців, які займаються глибоким навчанням [17,18].

Бібліотека вирізняється модульною структурою, що дозволяє легко налаштовувати та розширювати її функціонал. Keras є відкритим проєктом з безкоштовним вихідним кодом, доступним для перегляду та модифікації. Назва бібліотеки походить від грецького слова «ріг», що є відсилкою до міфологічних образів з «Одіссеї».

Основна сфера застосування Keras — машинне навчання на Python. Це ключовий інструмент для фахівців, які проєктують та тестують нейронні мережі й системи штучного інтелекту. Бібліотека надає простий і зрозумілий API, який допомагає швидко створювати й навчати моделі нейронних мереж. Keras підтримує різноманітні архітектури, включаючи згорткові (CNN) та рекурентні нейронні мережі (RNN), що дозволяє реалізовувати широкий спектр завдань у галузі машинного навчання.

Keras відзначається простотою у вивченні та використанні завдяки високорівневому Python-інтерфейсу, що забезпечує абстракцію складних обчислень. Хоча цей підхід робить Keras дещо повільнішим порівняно з іншими фреймворками глибокого навчання, він є ідеальним для новачків.

Бібліотека підтримує різні бекенди, зокрема TensorFlow, CNTK і Theano, що дозволяє користувачам обрати оптимальне середовище для своїх завдань. Далі переглянемо основні особливості бібліотеки:

– інтуїтивно зрозумілий інтерфейс для побудови моделей, що сприяє швидкому експериментуванню з архітектурами нейронних мереж;

– можливості для збільшення зображень, заповнення послідовностей та одноразового кодування;

– бібліотека працює зі згортковими мережами (CNN), рекурентними мережами (RNN), автокодерами та іншими архітектурами;

– Keras є частиною TensorFlow, що забезпечує потужну серверну підтримку для навчання нейронних мереж.

Бібліотека допомагає розробникам швидко й ефективно створювати застосунки на основі машинного навчання. Переваги Keras включають:

– скорочення коду завдяки його елегантності та лаконічності;

– спрощену ітерацію та легке налагодження моделей;

– прискорену роботу моделей через компіляцію XLA та оптимізацію Autograph;

– можливість розгортання на різних платформах: серверах, мобільних пристроях, браузерах та вбудованих системах завдяки TF Lite, TF.js і TF Serving.

Як діяти, якщо модель показує низьку точність? Якщо результат навчання моделі недостатній, можна спробувати такі кроки:

– триваліше навчання дозволяє алгоритму краще оптимізувати параметри та знайти ефективні рішення;

– недостатня кількість даних може стати причиною низької якості моделі, особливо якщо набір даних містить мало прикладів;

– для задач класифікації необхідно, щоб дані були рівномірно розподілені між усіма класами. Наприклад, якщо є 100 зразків на 4 класи, але перші два класи містять 90 зразків, а інші два — лише 10, це призведе до некоректних прогнозів для менш представлених класів.

1.2 Огляд та аналіз наявних аналогів та публікацій

В першій проаналізованій публікації [16] було використано Згорткові нейронні мережі (CNN), а саме VGG16 для визначення цілей на SAR-зображені. Синтетична апертурна радіолокація (SAR) володіє такими характеристиками, як всепогодність, всепроникність, далекодія та висока роздільна здатність, що робить

її важливим інструментом у галузях бойової розвідки, виявлення та наведення. Технологія розпізнавання цілей на основі SAR-зображень, особливо розпізнавання наземних військових цілей, привернула значну увагу. Набір даних MSTAR включає SAR-зображення нерухомих наземних цілей, наданих Агентством передових дослідницьких проектів оборони (DARPA) і Лабораторією досліджень ВПС (AFRL), і містить цивільні та військові об'єкти. Тож для виконання поставленої задачі, автори використовували згорткові нейронні мережі (CNN), що складаються з ряду згорткових, пулінгових і повнозв'язаних шарів, дозволяють ефективно представляти ознаки з великих даних та автоматично навчатися, спрощуючи процес витягу ознак і їх зіставлення. CNN вже широко застосовуються в інтерпретації цілей на основі SAR-зображень. Експерименти показали, що використання існуючої нейронної мережі VGG16 для класифікації військових цілей на наборі даних MSTAR дозволяє досягти високої точності класифікації.

Останніми роками безпілотні літальні апарати (БПЛА) стали незамінним інструментом у розвідувальних місіях (ISR), кардинально змінюючи підходи до військових операцій у різних конфліктах. Ці безпілотники забезпечують значні переваги — від прямої підтримки на полі бою до покращеного моніторингу місцевості та розуміння ситуації з противником. У даній публікації [17] представлено метод автоматичного виявлення цивільних і військових осіб на землі в режимі реального часу за допомогою стандартної RGB-камери, інтегрованої в БПЛА, та нейронної мережі для виявлення. Для навчання нейронної мережі було зібрано та розмічено набір даних, що дозволяє проводити навчання під наглядом. Запропонований підхід був експериментально перевірений, і фінальна модель показала обнадійливі результати, демонструючи великий потенціал для використання як інструмент підтримки ISR-місій.

Метою роботи в данній публікації [18] є огляд цих методів, аналіз їхніх алгоритмів, технік, наборів даних, роздільної здатності та типів зображень, а також обговорення їхніх сильних і слабких сторін. У епоху штучного інтелекту

дистанційне зондування, особливо супутникові знімки, стає все більш цікавим для спільноти комп'ютерних наук, оскільки це сприяє наданню машинам здатності розпізнавати навколишнє середовище через класифікацію супутникових зображень. Супутникові знімки Землі використовуються для збору, аналізу та обробки даних як у цивільних, так і у військових цілях. Вони мають численні застосування в таких галузях, як метеорологія, океанографія, рибальство, сільське господарство, біорізноманіття, геологія, картографія, планування землекористування та військові операції. Класифікація супутникових зображень перетворює ці зображення на корисну інформацію замість просто зображення місцевості. Вона базується на різних підходах і методах, які застосовуються залежно від конкретних обставин та умов. Ці методи можна поділити на п'ять категорій: керована класифікація, некерована класифікація, класифікація на основі пікселів, об’єктно-орієнтована класифікація та класифікація за допомогою згорткових нейронних мереж (CNN).

Коли мова іде про можливі варіанти архітектур нейромереж, що можна використати під час класифікації об’єктів на фото, то в такому випадку зазвичай виділяють архітектури, що описано нижче.

Прямі нейронні мережі — в даній архітектурі, інформація рухається лише в певному напрямі: від вхідного шару до вихідного. Ці мережі складаються з кількох шарів вузлів, і кожен вузол у певному шарі з'єднаний із вузлами попереднього та наступного шарів поміж зважені зв'язки. Всі вузли обробляють оримані від попереднього шару сигнали за допомогою активаційної функції, а результати обчислень на вихідному шарі представляють підсумковий результат роботи мережі [3].

Рекурентні нейронні мережі (RNN) мають здатність враховувати дані з попередніх кроків, еоли проводять обробку нового вхідного сигналу, завдяки чому інформація в них може рухатися в обох напрямках: вперед і назад,

використовуючи контекст минулих даних. У RNN існують особливі рекурентні вузли, що містять внутрішню пам'ять, що дає мжливість мережі кешувати та користуватись інформацією з минулих вхідних значень при аналізі поточних даних. Це забезпечує можливість працювати з послідовностями вхідних даних, враховуючи історичний контекст, що особливо корисно при роботі з часовими рядами, текстами та іншими послідовними даними.

Автокодувальні нейронні мережі (ACNN) — це вид нейромереж, який застосовується для стиснення та очищення вхідних даних, зберігши основну інформацію, необхідну для відновлення цих даних.

Основні складові автокодувальної мережі — кодер і декодер. Кодер обробляє вхідні дані, зменшуючи їх розмір за допомогою спеціалізованого шару нейронів, після чого цей стислий кодер відправляється на декодер для відтворення початкових даних.

У процесі тренування мережі використовуються вхідні дані, де мережа шукає оптимальний метод для стискання в кодері та відновлення в декодері. Часто застосовують техніки для зменшення розмірності та шуму даних.

Нейромережі з довільною архітектурою — це гнучкі інструменти, що дозволяють конструювати моделі без жорстких обмежень. На відміну від фіксованих структур персептронів чи згорток, вони можуть адаптуватися до будь-якої задачі, динамічно змінюючи кількість шарів та нейронів. Завдяки цій властивості, такі мережі успішно вирішують складні завдання, де традиційні підходи вичерпують себе, наприклад, розпізнавання комплексних образів або управління рухом роботів.

Глибинні нейронні мережі (DNN) — є системами, що збираються з багатьох шарів обчислювальних вузлів, де всі його шари виконують операції з даними. Ці мережі застосовуються для вирішення комплексних завдань у різноманітних

сферах, включаючи обробку зображень, розпізнавання мовлення, аналіз природних мов тощо [21, 22].

Характерною рисою глибинних нейронних мереж є те, що кожен наступний рівень використовує результати попереднього для створення складніших функцій. Це дозволяє мережі виявляти більш комплексні закономірності в даних.

Тренування глибинних нейронних мереж потребує значного обсягу даних та обчислювальних потужностей. Для оптимізації кількості параметрів та підвищення ефективності навчання можуть застосовуватися такі підходи, як згорткові та рекурентні нейронні мережі.

Нейронні мережі з підкріпленням — це особливий тип нейронних мереж, що використовуються для керування поведінкою агента в заданому середовищі.

Ці мережі базуються на алгоритмах навчання з підкріпленням, які ґрунтуються на принципі винагород та покарань. Агент діє в певному середовищі і приймає рішення з метою максимізації винагороди. Нейронна мережа з підкріпленням вдосконалюється шляхом взаємодії з середовищем та аналізу отриманих винагород.

Генеративні згорткові мережі — це тип нейромереж, які комбінують згорткові шари для створення нових зображень.

GCN переважно застосовують для генерації зображень через навчання без учителя, при якому мережа навчається розпізнавати основні риси зображення і відтворювати їх у нових кадрах [8].

Структурно GCN включають енкодер і декодер: енкодер стискає вхідне зображення до меншого набору ознак, а декодер перетворює їх у вихідний кадр. Це дозволяє генерувати нові, раніше не використані зображення.

Мережі для коротких відповідей — це тип нейронних мереж, розроблених для завдання обробки мови природи, наприклад, відповідь на запитання та взаємодія з користувачами.

SMRN включають два основні компоненти: енкодер і декодер. Енкодер отримує запит користувача і перетворює його на вектор прихованого стану. Декодер на основі цього вектора генерує коротку відповідь.

Завдяки здатності працювати з текстом на рівні слів і фраз, SMRN підходять для завдань розмовного пошуку, рекомендацій, відповідей на запитання, створення чат-ботів та інших задач, пов'язаних із обробкою природної мови.

Нейронні мережі з підтримкою пам’яті — це тип мереж, які можуть зберігати дані та отримувати до них доступ під час обробки вхідних сигналів. Вони застосовують пам’ять для зберігання контекстної інформації, що дозволяє ефективніше працювати з новими даними [5].

Одним із найбільш відомих прикладів таких мереж є NTM, запропонована в 2014 році. NTM містить механізм, що зберігає значні обсяги інформації, які можна використовувати для обробки природної мови, перекладу, розпізнавання образів та інших завдань, де критично важливий доступ до контексту.

1.3 Постановка задачі

Об'єкт роботи — процес класифікації військової техніки за допомогою нейронних мереж.

Предмет роботи — нейронні мережі для класифікації військової техніки на фотографіях.

Метою кваліфікаційної роботи є підвищення точності класифікації об'єктів на зображеннях за допомогою моделей нейронних мереж ResNet і DenseNet. Головні завдання дослідження включають:

– аналіз існуючих методів і алгоритмів класифікації зображень, зокрема нейронних мереж;

– вивчення теоретичної бази нейронних мереж DenseNet та ResNet, їх архітектурних особливостей і принципів роботи;

– збір і підготовка набору даних для класифікації зображень;

– навчання нейронної мережі ResNet на підготовлених даних та оцінка її ефективності;

– проведення експериментів для поліпшення результатів класифікації, зокрема із застосуванням технік, таких як transfer learning і fine-tuning;

– формулювання висновків щодо ефективності та потенціалу вдосконалення використаних нейронних мереж для класифікації об'єктів на фото та визначення перспектив подальших досліджень.

Висновки до розділу 1

Дослідження використання нейронних мереж для класифікації військових об'єктів на фото є надзвичайно актуальним у сучасному світі, де зростає кількість зображень, що вимагають автоматичної обробки та класифікації. Нейронні мережі, застосовані в цій роботі, належать до найбільш ефективних і часто використовуваних для класифікації зображень, тому вивчення їхніх моделей має велике значення для розвитку комп'ютерного зору та машинного навчання.

У першому розділі магістерської кваліфікаційної роботи розглянуто ключові поняття і визначення, пов’язані з дослідженням моделей нейронних мереж ResNet і DenseNet для класифікації військових об’єктів на фото. Проведено аналіз усіх аналогічних систем, що застосовуються для класифікації об'єктів на зображеннях, а також виявлено їхні переваги й недоліки.

Важливою задачею роботи є дослідження архітектурних можливостей нейромереж ResNet та DenseNet, які належать до найефективніших і найпоширеніших для класифікації зображень. Завдання полягає у дослідженні цих моделей з внесенням власних модифікацій на різних наборах даних і пошуку оптимальних параметрів мереж для отримання найвищої точності класифікації.

Завдяки проведеному аналізу і поставленій задачі, можна очікувати, що дослідження роботи нейронних мереж ResNet і DenseNet буде значущим та актуальним для розвитку комп’ютерного зору і машинного навчання. Отримані результати можуть стати корисними для вдосконалення систем класифікації зображень і знайдуть подальше застосування в різних сферах, зокрема в медицині, промисловості та безпеці.

2 НЕЙРОМЕРЕЖЕВІ ТЕХНОЛОГІЇ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ КЛАСИФІКАЦІЇ ВІЙСЬКОВОЇ ТЕХНІКИ НА ФОТО ТА ІНСТРУМЕНТИ СТВОРЕННЯ ВЕБ СИСТЕМИ РОЗПІЗНАВАННЯ

2.1 Архітектура нейромережі DenseNet

DenseNet (Dense Convolutional Network) — це архітектура нейронної мережі, яка покращує передавання інформації і градієнтів у глибоких мережах. Вона була запропонована для розв'язання проблеми зникнення градієнта та надмірної кількості параметрів у традиційних глибоких нейронних мережах. Далі перерахуємо основні ідеї та особливості DenseNet.

У DenseNet кожен шар має прямі з'єднання з усіма попередніми шарами. Це означає, що вхід кожного шару складається з виходів усіх попередніх шарів, а не тільки з попереднього шару, як у класичних архітектурах (наприклад, ResNet). Завдяки цьому мережа ефективно передає градієнти під час навчання, що полегшує оптимізацію.

Хоча DenseNet має багато зв'язків, кількість параметрів значно менша порівняно з іншими глибокими мережами. Це досягається тим, що мережа не вимагає великої кількості фільтрів у кожному шарі, оскільки інформація від попередніх шарів використовується безпосередньо.

DenseNet забезпечує ефективне повторне використання ознак, що дозволяє мережі вчитися більш ефективно, оскільки інформація з попередніх шарів не втрачається і може бути використана на подальших рівнях.

DenseNet будується з блоків щільного з'єднання (Dense Blocks), між якими знаходяться шари переходу (Transition Layers). Кожен блок щільного з'єднання складається з декількох шарів, які всі підключені один до одного. Шари переходу зменшують розмірність даних, що дозволяє контролювати розмір моделі.

Щільні з'єднання допомагають ефективніше передавати градієнти, зменшуючи проблему зникнення градієнта. Менша кількість параметрів робить модель більш ефективною щодо пам'яті та обчислювальної потужності. Завдяки зменшенню перенавчання, DenseNet часто демонструє кращі результати на багатьох завданнях розпізнавання образів.

Хоча DenseNet є ефективною за кількістю параметрів, вона може вимагати більше обчислювальних ресурсів через велику кількість з'єднань. Крім того, реалізація щільних з'єднань може бути складнішою і вимагає більшої пам'яті під час тренування.

DenseNet добре підходить для задач класифікації зображень, сегментації, розпізнавання об'єктів та інших застосувань у комп'ютерному зорі. Вона використовується в багатьох сучасних рішеннях завдяки її ефективності та високій точності.

ResNet суттєво змінив уявлення про те, як параметризувати функції в глибоких нейронних мережах. DenseNet, певною мірою, є логічним продовженням цього. Щоб зрозуміти, як до нього дійти, давайте зробимо невеликий теоретичний відступ. Нагадаємо розклад Тейлора для функцій. Для скалярів він може бути записаний як:

$$f(x) = f(0) + x * \left[f'(0) + x * \left[\frac{f''(0)}{2!} + x * \left[\frac{f'''(0)}{3!} + \ \ldots \ \right] \right] \right]$$

Основний момент полягає в тому, що він розкладає функцію на терміни з дедалі вищим порядком. Аналогічно, ResNet розкладає функції на:

$$f(x) = x + g(x)$$

Тобто ResNet розкладає f на простий лінійний термін і більш складний нелінійний. Що, якби ми хотіли врахувати (не обов’язково додати) інформацію за межами двох термінів? Одне з таких рішень — DenseNet.

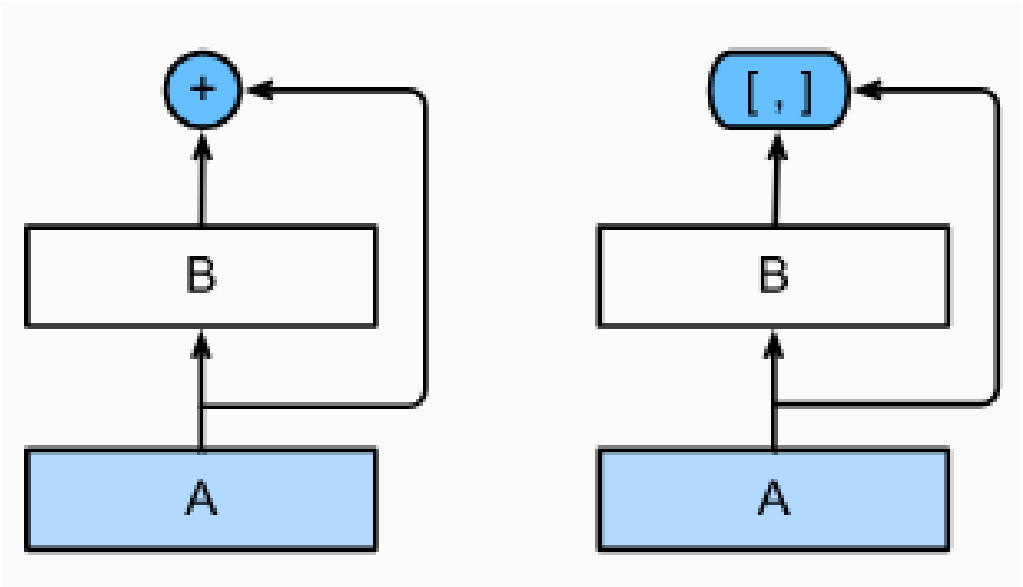

Рисунок 2.2 — Схеми роботи ResNet і DenseNet

Як показано на малюнках, ключова відмінність між ResNet і DenseNet полягає в тому, що у другому випадку виходи *конкатенуються* (позначається [,]) (рис. 2.2), а не додаються. У результаті ми виконуємо відображення від x до його значень після застосування дедалі складнішої послідовності функцій (рис. 2.3):

$$x \to \left[x,\ f_1(x),\ f_2([x,\ f_1(x)]),\ f_3(x,\ f_1(x),\ f_2([x,\ f_1(x)])),\ \ldots\ \right]$$

У підсумку всі ці функції комбінуються в багатошаровому персептроні (MLP), щоб знову зменшити кількість ознак. З точки зору реалізації це досить просто: замість додавання термінів ми конкатенуємо їх.

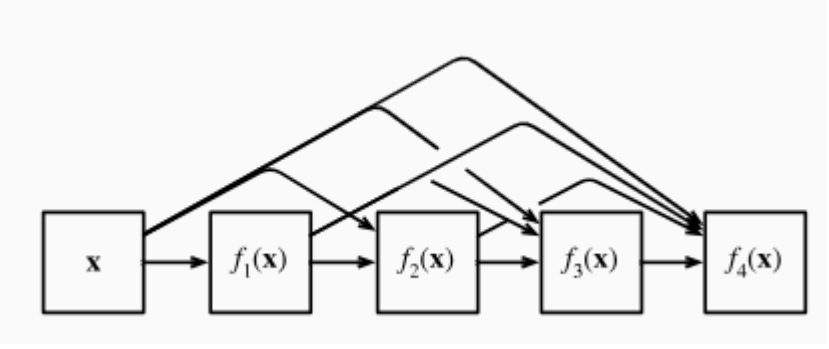

Рисунок 2.3 — Схема роботи блоків в моделі DenseNet

Основними компонентами, що складають DenseNet, є *щільні блоки* (dense blocks) та *перехідні шари* (transition layers) (рис. 2.4). Перші визначають, як вхідні та вихідні дані конкатенуються, тоді як другі контролюють кількість каналів, щоб вона не була занадто великою, оскільки розширення може бути дуже високорозмірним.

Архітектура повнозгорткової нейронної мережі DenseNet починається з вхідного шару розміром (224, 224, 3). Далі застосовується первинна згорткова операція за допомогою шару Conv2D з 24 фільтрами та ядром розміром 3x3. Після цього йде активація функцією ReLU, а потім шар MaxPooling2D з вікном 2x2 для зменшення розміру вхідних даних. Щоб стабілізувати навчання, застосовується шар BatchNormalization.

Після цього йдуть блоки повторюваних операцій, кожен з яких складається з шару Conv2D з 4 фільтрами, шару Dropout для регуляризації і шару Concatenate, який поєднує вихід цього блоку з результатами попередніх блоків. Ці блоки повторюються кілька разів, при цьому кількість фільтрів у кожному новому блоці збільшується на 4, що дозволяє моделі поступово будувати складніші представлення ознак.

Фінальна частина архітектури включає шар Conv2D з 88 фільтрами, після чого йде AveragePooling2D для зменшення розміру простору ознак до 2x2. Далі застосовується шар BatchNormalization, щоб нормалізувати вихідні дані перед застосуванням GlobalAveragePooling2D, що зменшує просторовий розмір до єдиного значення на кожен фільтр. Останній шар — це Dense з 3 вихідними нейронами, який використовується для класифікації на 3 класи.

Ця архітектура використовує повторювані згорткові блоки з Dropout та Concatenate, що дозволяє мережі глибоко обробляти вхідні дані та забезпечує складну побудову ознак на кожному рівні мережі.

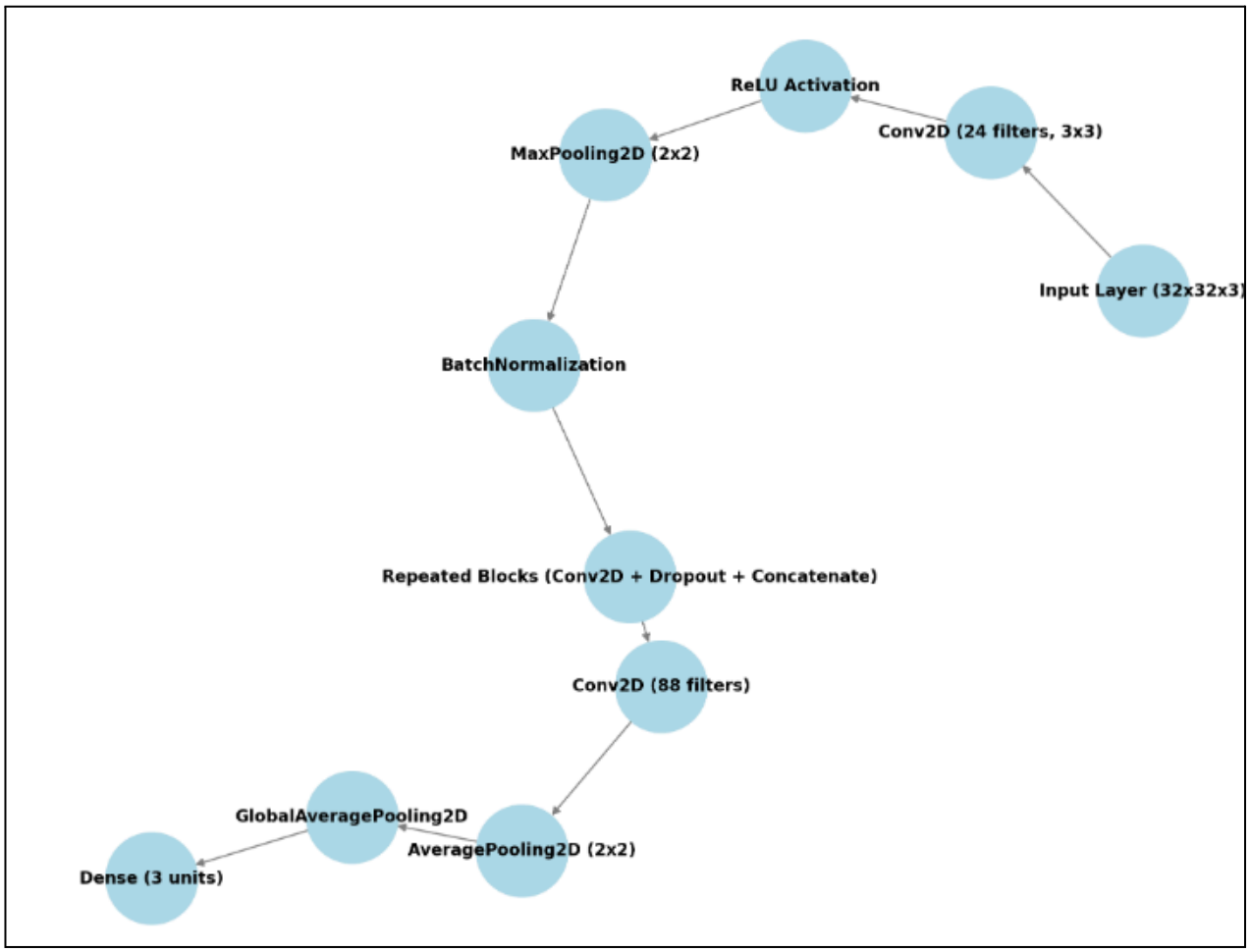

Рисунок 2.4 — Архітектура моделі нейромережі DenseNet

2.2 Архітектура нейромережі ResNet-50

ResNet-50 — це різновид архітектури ResNet, що складається з 48 згорткових шарів, а також по одному шару Max Pooling і Average Pooling. Модель виконує $3{,}8 \times 10^9$ операцій з плаваючою комою та є однією з найпоширеніших версій ResNet [6, 8].

ResNet-50 — це глибока нейронна мережа з 50 рівнями, основою якої є залишкові блоки (*residual blocks*), що поєднують кілька шарів для ефективного навчання. Весь список шарів, котрі входять до архітектури ResNet-50, наведено нижче:

– вхідний шар, представляє собою фото, що передається на класифікацію;

– перший convolution layer, що робить початкову обробку фото;

– блок, що складається зі зв'язків residual та зібраний з двох convolution layers і шару об’єднання;

– накступним кроком, повторюється пункт 2 та 3 ще тричі, що дозволить створити чотири residual блоки;

– після кожного блоку residual, використовуємо макс-пулінг, це дозволить зменшити розмір фото та концентрує увагу на ключових деталях;

– за останнім residual блоком, ще раз йде макс-пулінг блок;

– FC layer є останнім шаром, який відповідає за класифікацію зображення на певні категорії.

Блок residual — це головний елемент архітектури ResNet-50, який допомагає ефективно боротися з проблемою згасання градієнтів під час навчання [7]. У глибоких мережах згасання градієнтів може призвести до значного уповільнення або навіть зупинки навчання, оскільки з часом градієнти стають надто малими, аби коректно оновлювати параметри мережі.

Щоб уникнути цього, в ResNet застосовуються залишкові зв’язки, які створюють «шлях» для передачі інформації, минаючи декілька шарів. Це дозволяє зберігати й передавати важливі дані через всю мережу без значного зменшення сигналу. Замість прямого передавання сигналу від першого шару до другого, залишковий блок додає оброблений сигнал до вихідного, забезпечуючи, таким чином, більш ефективне перенесення інформації далі в мережі.

Спочатку ResNet розроблялася для класифікації об’єктів на фото, але її ефективність дала змогу використовувати архітектуру і в інших галузях, що ніяким чином не зв’язані з computer vision, де вона також показала високу точність. Можна подумати, що додавання шарів теж могло б підвищити точність,

проте для тренування надглибоких мереж все ж потрібно залишкове навчання [5, 6].

Архітектура ResNet ефективна для завдань комп'ютерного зору, таких як класифікація зображень, локалізація об’єктів і виявлення об’єктів. Вона також може застосовуватись для задач, не пов’язаних з комп'ютерним баченням, щоб скористатися перевагами глибини моделі й знизити обчислювальні витрати.

Однією з основних проблем глибоких нейронних мереж є зникнення або вибух градієнтів. Різні методи допомогли подолати цю складність і дозволили тренувати мережі з десятками шарів. Проте зі збільшенням глибини моделей виникла інша серйозна проблема: насичення точності, після чого її значення починає різко знижуватися. Цікаво, що це явище не пов’язане з перенавчанням, оскільки додавання шарів до моделі лише збільшувало помилку навчання.

Для вирішення цієї проблеми застосували підхід, у якому глибока модель будувалася на основі дрібної моделі, до якої додавалися шари ідентичності. Завдяки такій архітектурі, глибша модель не демонструвала вищу помилку навчання порівняно зі своєю дрібнішою версією, оскільки нові шари не вносили додаткових змін до вихідних даних.

У ResNet-50 та новіших версіях було впроваджено кілька важливих змін:

– швидкі з’єднання (skip connections) тепер охоплюють три шари, а не два, як раніше;

– додано шари згортки 1x1 для оптимізації обчислень і зменшення кількості параметрів.

Далі ми детально розглянемо структуру ResNet-50 та особливості цих змін (рис. 2.5).

Перший шар згорткової нейронної мережі реалізується за допомогою згортки з ядром розміром 7×7 і 64 каналами. Усі ядра цього шару мають крок (stride) 2, що дозволяє значно зменшити просторові розміри вхідних даних. Після

цієї операції йде шар максимального об'єднання (max pooling) з кроком 2, який ще більше знижує розміри просторового представлення.

Далі, на наступному етапі, використовується комбінація трьох послідовних згорткових шарів:

- початкове ядро має розмір 1×1 із 64 каналами, яке відповідає за зменшення кількості каналів;
- друге — ядро розміром 3×3, також із 64 каналами, яке виконує просторову згортку;
- третє ядро розміром 1×1 із 256 каналами збільшує кількість каналів після обчислень.

Ця комбінація трьох шарів повторюється тричі, формуючи загалом 9 шарів у цьому блоці.

Наступний блок побудований за аналогічним принципом, але з іншими характеристиками ядер:

- спочатку використовується ядро 1×1 із 128 каналами;
- потім ядро 3×3, також із 128 каналами;
- нарешті, ядро 1×1 із 512 каналами.

Цей блок повторюється чотири рази, що додає ще 12 шарів до мережі.

На третьому етапі конфігурація ядер змінюється:

- перше ядро має розмір 1×1 із 256 каналами;
- друге — 3×3, також із 256 каналами;
- третє ядро 1×1 працює з 1024 каналами.

Ця структура повторюється шість разів, додаючи 18 шарів до моделі.

Четвертий блок має ще більшу кількість каналів:

- ядро 1×1 із 512 каналами;
- ядро 3×3, також із 512 каналами;
- ядро 1×1 із 2048 каналами.

Ця структура повторюється три рази, додаючи до моделі ще 9 шарів.

Наприкінці моделі використовується середній пулінг (average pooling), що підсумовує просторову інформацію, після чого додається повністю підключений шар із 1000 вузлами, що відповідають за класифікацію на 1000 категорій. Завершальною функцією є softmax, яка перетворює вихідні значення у ймовірності.

Якщо не враховувати функції активації та операції об'єднання (пулінгу), загальна кількість шарів у моделі становить:

$$1 + 9 + 12 + 18 + 9 + 1 = 50$$

Ця архітектура формує глибоку згорткову нейронну мережу з оцінкою максимальної обчислювальної потужності в 3.8109 FLOPs (floating-point operations per second).

layer name	output size	50-layer	101-layer
conv1	112×112	7×7, 64, stride 2	
conv2_x	56×56	3×3 max pool, stride 2	
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$
conv4_x	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$
conv5_x	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax	
FLOPs		3.8×10^9	7.6×10^9

Рисунок 2.5 — Порівняльна таблиця архітектур моделей ResNet

Floating point operations per second (FLOPS) — основне позначення кількісті операцій з плаваючою комою, які можуть бути виконаними за секунду. Також часто застосовується підчас оцінки обчислювальної потужності апаратних засобів, таких як процесори (CPU) або графічні процесори (GPU) [2].

2.3 Архітектура нейромережі ResNet-101

ResNet-101 — це глибока нейронна мережа, представлена у 2015 році в дослідженні "Deep Residual Learning for Image Recognition", авторами якого є Kaiming He, Xiangyu Zhang, Shaoqing Ren та Jian Sun з Microsoft Research Asia.

Ця архітектура є частиною сімейства ResNet і використовує залишкові з'єднання (residual connections), що дозволяють ефективно розв'язувати проблему зникнення градієнтів у дуже глибоких нейронних мережах. Завдяки цьому ResNet-101 досягає високої продуктивності та стабільного навчання навіть при великій кількості шарів.

ResNet-101 має 101 шар, що робить їїархітектуру глибшою, порівняно з ResNet-50. Архітектура складається з двох ключових компонентів: "базового блоку" і "бутель-блоку". Базовий блок включає пару згорткових шарів, шар нормалізації (Batch Normalization) і "residual connection", що сприяє ефективній передачі інформації. У свою чергу, бутель-блок об’єднує три таких базових блоки, зменшуючи розмір зображення вдвічі та збільшуючи кількість каналів.

ResNet-101 є складною і ресурсомісткою моделлю, що потребує значної обчислювальної потужності для навчання та роботи, і зазвичай використовується для великих задач класифікації зображень, як-от ImageNet [6].

Архітектура ResNet-101 є однією з найпопулярніших глибоких згорткових нейронних мереж, що забезпечує ефективне навчання завдяки використанню спеціальних блоків із пропусканням залишкових зв’язків (residual connections).

Мережа починається з початкового шару згортки з ядром розміром 7×7 і 64 фільтрами, які виконуються з кроком 2. Цей шар дозволяє зменшити просторові розміри вхідного зображення, знижуючи обчислювальну складність наступних шарів. За ним йде шар максимального об'єднання (max pooling) з кроком 2, який додатково зменшує розмірність і зберігає найважливішу інформацію для подальшої обробки.

Далі архітектура використовує комбінацію шарів згортки із ядрами різних розмірів (1×1, 3×3) і кількістю каналів, що поступово збільшується. У першій частині моделі йдуть блоки з ядрами 1×1, 3×3, 1×1, які мають 64, 64, і 256 каналів відповідно. Ця комбінація повторюється тричі, додаючи до моделі 9 шарів.

Наступні блоки включають згорткові шари з ядрами 1×1, 3×3, 1×1, із каналами 128, 128, і 512 відповідно. Цей набір повторюється чотири рази, додаючи 12 шарів.

Далі використовуються шари із ядрами 1×1, 3×3, 1×1, які мають 256, 256, і 1024 каналів відповідно. Цей набір повторюється 23 рази, забезпечуючи додаткові 69 шарів у моделі.

Останні блоки містять шари з ядрами 1×1, 3×3, 1×1 із каналами 512, 512, і 2048. Цей набір повторюється тричі, додаючи ще 9 шарів.

На завершення використовується середній пулінг (average pooling), що зводить просторову інформацію до одного значення на канал. Потім додається повністю підключений шар із 1000 вузлами, що відповідає за класифікацію, а виходи обробляються функцією softmax для отримання ймовірностей.

Підсумовуючи, загальна кількість шарів у моделі ResNet-101 дорівнює:

$$1 + 9 + 12 + 69 + 9 + 1 = 101$$

Ця архітектура забезпечує обчислювальну потужність на рівні 7.6109 FLOPs, що дозволяє ефективно обробляти складні завдання комп'ютерного зору, такі як класифікація, детекція та сегментація.

Після встановлення обчислювальних можливостей нейронної мережі ResNet розглянемо її функціонування крок за кроком [4].

1. Спочатку на вхід мережі подається вихідне зображення. Першим етапом є конволюційний шар, який здійснює початкову обробку зображення.

2. Далі використовуються послідовно розташовані блоки з "residual" зв'язками. Кожен блок складається з двох конволюційних шарів та шару додавання. Цей додатковий шар додає початковий вхід до вихідного сигналу конволюції, зберігаючи важливу інформацію, отриману на попередньому етапі обробки.

3. Після виконання обробки "residual" блоками здійснюється макс-пулінг, який зменшує розмір зображення, акцентуючи увагу на основних рисах. Потім повторюється та ж послідовність: блоки "residual" і макс-пулінг. У ResNet-50 цей процес повторюється ще тричі.

4. На завершальному етапі модель переходить до повнозв'язного шару, що класифікує зображення по відповідних категоріях. У результаті на виході формується перелік ймовірностей, що вказують на можливість приналежності об'єкта на фото до кожної з категорій.

2.4 Нейромережі, придатні для дослідження

Іншим важливим варіантом архітектури є VGG, названа на честь дослідницької групи з Оксфордського університету, яка спеціалізується на комп'ютерному зорі та аналізі зображень.

Архітектура VGG (рис. 2.6) складається з кількох модулів, що представляють собою серію згорткових шарів, які чергуються з шарами пулінгу.

Кількість згорткових і пулінгових шарів у кожному модулі є фіксованою, що дозволяє легко адаптувати глибину мережі відповідно до потреб. Незважаючи на свою високу потужність, завдяки великій кількості параметрів, VGG потребує значних обчислювальних ресурсів як для навчання, так і для використання.

VGG — популярна модель для класифікації зображень, яка знайшла широке застосування в різних задачах комп'ютерного зору, таких як розпізнавання облич, детекція об'єктів та інші. Вона стала основою для численних досліджень і модифікацій, серед яких VGG16 та VGG19 є найбільш відомими версіями.

VGG16 і VGG19 — це глибокі нейронні мережі, що забезпечують високу продуктивність у завданнях класифікації зображень. Вони відіграють важливу роль у розвитку архітектур глибокого навчання та широко використовуються як базові моделі для подальших розробок.

Рисунок 2.6 — Архітектура VGG16.

Архітектура VGG16 включає в себе 16 шарів, з яких 13 представляють згорткові шари, а 3 — повнозв’язні. Згорткові шари використовують фільтри різного розміру, щоб обробляти зображення та виділяти різноманітні ознаки, необхідні для класифікації. Повнозв’язні шари приймають ці ознаки для подальшого розподілу зображень по відповідним категоріям.

Архітектури VGG16 та VGG19 мають однакову архутектуру, за вийнятком наявності ще 3 додаткових згорткових шарів у останньї. Це дає покращену точність класифікації об’єктів, але через це ма є збільшену потребу в обчислювальних потужностях для тренування та використання.

VGG19 та VGG16, завдяки їхній високій точності та простоті у реалізації, знайшли своїх послідовників для задач класифікації зображень. Також використовуються як основні моделі в багатьох інших завданнях глибинного навчання.

Для розв'язання задачі магістерської кваліфікаційної роботи, за умови наявності достатніх обчислювальних ресурсів, доцільно використовувати глибші та складніші архітектури нейронних мереж, такі як ResNet і VGG, оскільки вони зазвичай демонструють вищу точність у порівнянні з AlexNet.

Архітектура VGG відзначається більшою кількістю шарів, що дозволяє досягти високої точності класифікації зображень завдяки ускладненню моделі. У свою чергу, ResNet вирізняється застосуванням залишкових блоків (*residual connections*), які дають змогу будувати дуже глибокі мережі з меншою кількістю параметрів. Це робить ResNet ефективною та популярною моделлю для різноманітних завдань у галузі комп'ютерного зору.

AlexNet — це одна з перших згорткових нейронних мереж, яка продемонструвала значні переваги глибокого навчання у задачах обробки та класифікації зображень. Вона стала проривом у сфері комп’ютерного зору завдяки своїй здатності досягати високої точності при класифікації складних зображень. Назва цієї архітектури походить від імені її основного автора, Алекса Кріжевського, який разом із Джеффрі Гінтоном та Іллею Суцкевером розробив її у 2012 році. AlexNet стала відомою завдяки перемозі на конкурсі ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2012), де вона суттєво перевершила всі існуючі на той час методи.

Архітектура AlexNet складається з 5 згорткових шарів, які є основою її здатності обробляти зображення. Перші два згорткові шари оснащені великою кількістю фільтрів, що дозволяє моделі виявляти різноманітні особливості зображень — від простих, таких як краї, до більш складних, які складаються із комбінацій ознак. Кожен згортковий шар супроводжується шаром пулінгу (переважно max-pooling), який виконує дві основні функції: зменшує розмір просторових даних і знижує обчислювальні витрати. Це також сприяє зменшенню ймовірності перенавчання завдяки агрегації інформації з сусідніх пікселів.

Крім того, AlexNet використовує ReLU (Rectified Linear Unit) (рис. 2.7) як функцію активації після кожного шару, що дозволяє уникати проблеми затухання градієнта, характерної для старіших активаційних функцій, таких як сигмоїдна. Архітектура також включає два повністю зв’язані шари в кінці, які відповідають за остаточну класифікацію. Для боротьби з перенавчанням мережа використовує техніку drop-out, яка випадково вимикає нейрони під час тренування, покращуючи її узагальнюючу здатність.

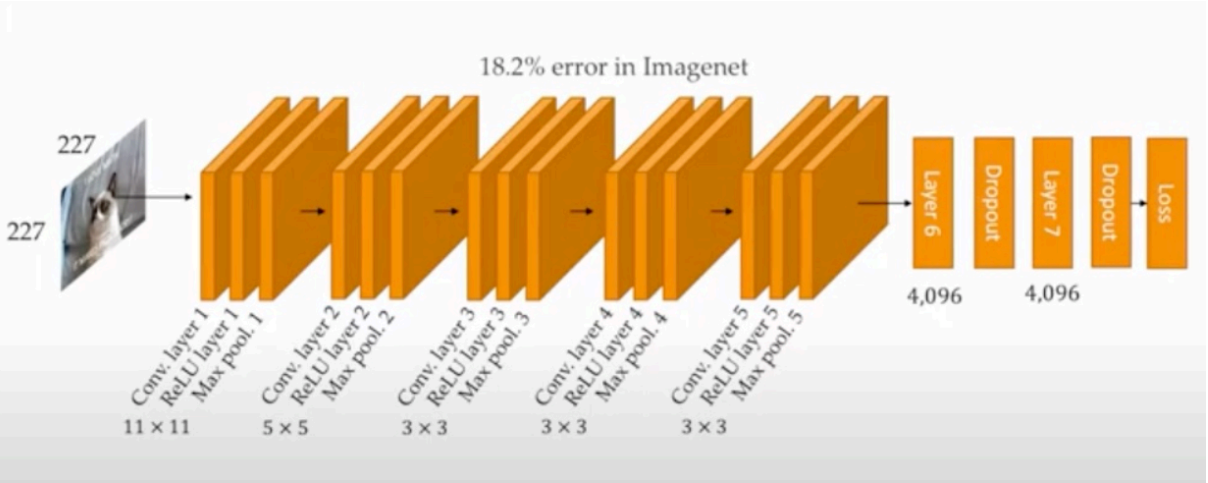

Рисунок 2.7 — Архітектура AlexNet

Після п’ятого згорткового шару AlexNet містить три повнозв’язні шари, кожен з яких складається з 4096 нейронів. Ці шари відповідають за класифікацію ознак, сформованих на попередніх етапах обробки.

Важливою особливістю AlexNet є застосування техніки Dropout. Вона полягає у випадковому відключенні певних нейронів під час тренування, що знижує ризик перенавчання та підвищує загальну ефективність моделі.

Для більш детального обгрунтування обраних нейромереж, створимо порівняльні таблиці для таких нейромереж як ResNet-50, ResNet101, AlexNet, VGG19, DenseNet.

Таблиця 1.1 — Порівняльна таблиця архітектур

Архітектура	ResNet-50	AlexNet	VGG19
Рік анонсу	2015	2012	2014
Глибина моделі	50 шарів	8 шарів	19 шарів
Головна інновація	Залишкові з'єднання (пропуск з'єднань)	Раннє використання ReLU та Dropout	Глибша мережа з використанням невеликих фільтрів
Кількість параметрів	~25.6M	~60M	~143.7M
Розміри ядра	В більшості 3×3 ядра	11×11, 5×5, 3×3	Розміри ядра
Компроміс між глибиною та шириною	Дуже глибокі, вузькі шари	Дрібні, широкі шари	Компроміс між глибиною та шириною
Обчислювальна ефективність	Високий через залишки	Помірний	Обчислювальна ефективність

Таблиця 1.2 — Порівняльна таблиця продуктивності

Метрики	ResNet-50	AlexNet	VGG19
FLOPs (Floating Point Operations)	~4.1 GFLOPs	~0.72 GFLOPs	~19.6 GFLOPs
Час висновку	Помірний	Швидко	Повільно
Час тренування	Довго	Швидко	Дуже довго

Продовження таблиці 1.2

Метрики	ResNet-50	AlexNet	VGG19
Використання памяті	Помірне	Низьке	Високе
Потреби GPU	Сучасні графічні процесори (16 ГБ+)	Графічні процесори нижчого класу	Графічні процесори високого класу (24 ГБ+)

Таблиця 1.3 — Порівняльна таблиця переваг та недоліків

Аспект	ResNet-50	AlexNet	VGG19
Переваги	Працює з дуже глибокими мережами; зменшує проблему зникання градієнтів	Простота та ефективність; перший, хто популяризував згорткові нейронні мережі (CNN)	Висока точність для свого часу; однорідна архітектура
Недоліки	Складний у впровадженні; потребує більше ресурсів	Схильний до перенавчання на менших наборах даних; відносно не глибокий	Вимагає надмірно багато пам’яті та обчислювальних ресурсів

У ході цього дослідження було оцінено продуктивність п'яти нейронних мереж у завданнях класифікації зображень. Основна увага приділялася порівнянню моделей за точністю з використанням набору даних ImageNet, який є стандартним еталоном для таких досліджень. Серед вивчених мереж модель ResNet-50 стабільно демонструвала найкращі результати, за нею йшли ResNet-101 та DenseNet. Для більш широкого порівняння також було включено ResNet-34 та AlexNet.

Нижче в таблиці 1.4 виконано порівняльний аналіз тестової перевірки роботоздатності моделей на тестовому датасеті CIFAR-10, котрий містить в собі 10000 фото розділених на 10 класів, що демонструє точність цих моделей.

Таблиця 1.4 — Порівняльна таблиця продуктивності

Model	Architecture Depth	Accuracy (%)		Dataset
		Top-1	Top-5	
ResNet-50	50 layers	76.2	92.8	CIFAR- 10
ResNet-101	101 layers	77.4	93.5	
DenseNet	88 layers	75.6	92.3	
ResNet-34	34 layers	74.6	91.8	
AlexNet	8 layers	56.5	79.1	

Представлені результати показують, що найвищу Top-1 точність демонструє ResNet-101 (77.4%), що очікувано, враховуючи її глибину в 101 шар. ResNet-50 із 50 шарами займає друге місце з точністю 76.2%, поступаючись ResNet-101, але випереджаючи DenseNet (75.6%), яка використовує 88 шарів. ResNet-34 із глибиною 34 шари демонструє точність 74.6%, що є нижчим через меншу складність архітектури. AlexNet із лише 8 шарами значно поступається всім іншим моделям із Top-1 точністю 56.5%, що підкреслює її обмеженість для сучасних задач. Аналіз Top-5 точності також підтверджує ці спостереження: ResNet-101 досягає 93.5%, ResNet-50 — 92.8%, а DenseNet — 92.3%, тоді як ResNet-34 і AlexNet демонструють 91.8% і 79.1% відповідно. Загалом результати вказують на те, що збільшення глибини моделі, як правило, покращує її точність, хоча ефективність також залежить від архітектурних особливостей.

2.5 Інструменти та технології для створення веб системи для розпізнавання з використанням обраних неромереж

На сьогоднішній день IT сфера пропонує розробникам величезну кількість різноманітних технологій, мов програмування та фреймворків для розробки веб системи, такі як: PhP, React, Laravel, Python, JavaScript, TypeScript, Vue.js та інші. Переглянувши відомості та дані по всім можливим технологіям було вирішено

розробити веб систему з використанням двох фреймворків, а саме: Angular — як фреймворк для клієнтської частини та ASP.NET MVC — для серверної сторони системи.

Angular — це веб-фреймворк з відкритим вихідним кодом, який підтримується Google і спільнотою розробників. Його створено для ефективної розробки динамічних та інтерактивних односторінкових додатків (SPA). З Angular розробники можуть створювати надійні, масштабовані та легкі у підтримці веб-додатки.

Angular, вперше випущений у 2010 році компанією Google, зазнав значних змін за ці роки. Перша версія, AngularJS, запровадила такі концепції, як двостороннє прив'язування даних і директиви. Однак зі зміною вимог до веб-розробки AngularJS почав стикатися з обмеженнями у продуктивності та гнучкості.

У 2016 році була випущена Angular 2 — повністю переписана версія AngularJS, яка зосередилася на модульності та продуктивності. З того часу Angular продовжує розвиватися, регулярно оновлюється та вдосконалюється, щоб відповідати сучасним вимогам веб-розробки.

Ось основні причини, чому Angular було обрано як фреймворк для клієнтської частини веб-системи:

– кастомні компоненти — Angular дозволяє створювати власні компоненти, які об'єднують функціонал і логіку відображення в повторно використовувані елементи;

– прив'язування даних — Angular дозволяє легко передавати дані з JavaScript-коду до інтерфейсу та реагувати на дії користувачів без необхідності вручну писати код;

– ін'єкція залежностей — Angular забезпечує написання модульних сервісів і їх інтеграцію в будь-яке місце, де вони потрібні. Це покращує тестованість і повторне використання сервісів;

– тестування — тестування є важливим аспектом, і Angular розроблений з урахуванням цього. Ви зможете тестувати всі частини вашого додатку, що настійно рекомендується;

– комплексність — Angular є повноцінним JavaScript-фреймворком, який пропонує готові рішення для серверної взаємодії, маршрутизації всередині додатку та інших завдань;

– сумісність із браузерами — Angular працює кросплатформено та сумісний із багатьма браузерами. Додатки на Angular зазвичай працюють у всіх основних браузерах (наприклад, Chrome, Firefox) та операційних системах, таких як Windows, macOS та Linux.

Angular-компоненти це невеликі частини інтерфейсу користувача, подібні до секцій у додатку. Хоча кожен компонент має власну функціональність, в Angular має сувору ієрархію формування компонентів.

Компонентне дерево (рис. 2.8) відображає структуру елементів інтерфейсу додатка.

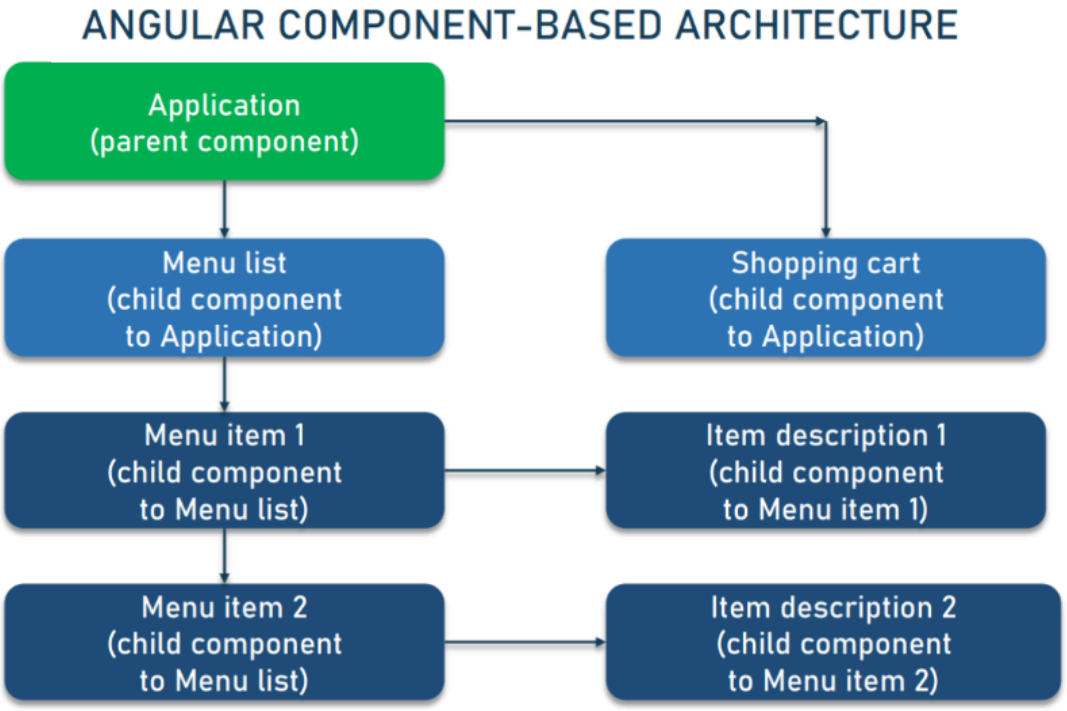

Рисунок 2.8 - Приклад схеми взаємодії компонентів в Angular

У наведеному прикладі компоненти є секціями eCommerce-додатка: сам додаток є батьківським компонентом, який містить список меню та кошик для покупок. Відповідно, список меню має дочірні компоненти, а кожен елемент меню також містить компонент з описом товару. Це і є ієрархія компонентів.

Компонентно-орієнтована архітектура дозволяє створювати інтерфейси з багатьма рухомими частинами та одночасно спрощує процес розробки для інженерів. Інші основні переваги такої архітектури:

– повторне використання — компоненти подібного типу добре інкапсульовані, тобто є самодостатніми. Розробники можуть повторно використовувати їх у різних частинах програми. Це особливо корисно у великих проєктах, де різні системи можуть мати багато схожих елементів, таких як пошукові поля, вибір дати, списки сортування тощо;

– читабельність — інкапсуляція також забезпечує кращу читабельність коду для нових розробників, які щойно долучилися до проєкту, що дозволяє їм швидше досягти продуктивності;

– зручність модульного тестування — незалежна природа компонентів спрощує модульне тестування та забезпечує якісну перевірку продуктивності найменших частин додатка.

У табл. 1.5, для наглядності та розуміння всієї інформації про Angular, наведено порівняння всих переваг та недоліків.

Таблиця 1.5 — Порівняння переваг та недоліків Angular

Переваги	**Недоліки**
Продуктивність — широкий набір інструментів та екосистема Angular спрощують завдання розробки, що дозволяє швидше завершувати проєкти.	Крива навчання — Angular має стрімку криву навчання, особливо для початківців, через складність концепцій і велику кількість інформації

Продовження таблиці 1.5

Переваги	**Недоліки**
Масштабованість — Angular чудово підходить для створення масштабних додатків завдяки своїй компонентній архітектурі та високій продуктивності.	Розмір — додатки Angular зазвичай мають більші розміри файлів у порівнянні з іншими фреймворками.
Підтримуваність — модульна архітектура Angular і чіткий поділ відповідальностей сприяють організації коду та його легкій підтримці.	Навантаження на продуктивність — потужні функції Angular мають свою ціну — погано оптимізовані додатки можуть стикатися з проблемами продуктивності.
Підтримка спільноти — завдяки підтримці Google та широкій спільноті розробників Angular користується сильною підтримкою та постійно вдосконалюється.	Міграція — оновлення між основними версіями Angular може бути складним і тривалим процесом, який вимагає значних змін у наявному коді.

Angular є потужним і функціонально насиченим фреймворком для створення сучасних веб-додатків. Завдяки своїм можливостям, широким інструментам і сильній підтримці спільноти Angular залишається популярним вибором для розробників, які прагнуть створювати масштабовані та підтримувані SPA-додатки.

Незважаючи на криву навчання та проблеми продуктивності, Angular пропонує величезний потенціал для створення динамічного та інтерактивного веб-досвіду.

Незалежно від того, чи ви досвідчений розробник, чи новачок у веб-розробці, Angular надає інструменти та можливості для втілення ваших ідей у життя. Дивлячись на порівняльну таблицю (табл. 1.1) та беручи до уваги ті потреби, котрі повинен задовільняти фреймворк для клієнтської частини нашої системи, можна з упевненістю сказати, що недоліки Angular є незначущими для нашої системи, наведміну від всих його переваг, що і робить Angular підходящою для нас технологією.

При створенні серверної частини застосунку важливо спершу визначити його основні цілі та завдання, які він має виконувати. Наступним кроком є визначення функціональних вимог до вебзастосунку та ключових характеристик, таких як обсяг даних, необхідних для зберігання, підтримка завантаження файлів, методи взаємодії з користувачами тощо.

Особливе значення в процесі проєктування має розробка інтерфейсу користувача. Інтерфейс повинен бути інтуїтивно зрозумілим, зручним і відповідати очікуванням цільової аудиторії. Під час роботи над інтерфейсом важливо брати до уваги не лише естетичні аспекти, але й принципи зручності використання та доступності.

Після завершення етапу концептуального проєктування, функціонального планування та створення інтерфейсу слід визначити технічні вимоги до вебзастосунку, включаючи вибір мов програмування, баз даних, вебсерверів тощо [18].

Вебзастосунок — це програмне забезпечення, яке працює через веббраузер і доступне в Інтернеті. Такі застосунки розробляють для різних потреб, наприклад, електронної комерції, онлайн-банкінгу, соціальних мереж, ігор тощо.

Основними компонентами вебзастосунку є вебсервер, база даних і клієнтський інтерфейс. Вебсервер обробляє запити клієнтів та забезпечує зв’язок між ними та

сервером. База даних зберігає інформацію, яку використовує застосунок, а клієнтський інтерфейс слугує засобом взаємодії користувача з вебзастосунком [13].

Розробка вебзастосунку складається з кількох основних етапів: проєктування, програмування, тестування та розгортання. На етапі проєктування створюються макети інтерфейсу та визначаються функції застосунку. Розробка охоплює програмування серверної та клієнтської частин. Тестування включає перевірку функціональності, безпеки та продуктивності, а розгортання полягає в установці застосунку на сервер і його налаштуванні для роботи в Інтернеті.

ASP.NET MVC (Model-View-Controller) — це фреймворк від Microsoft для розробки веб-додатків, що базується на архітектурі MVC, яка розділяє логіку на три частини: Model (відповідає за дані та бізнес-логіку), View (відображає дані) та Controller (керує запитами й взаємодією між Model і View). Така структура забезпечує чітке розділення обов’язків, спрощує розробку та підтримку, а також сприяє масштабованості. ASP.NET MVC підтримує Razor-синтаксис для створення динамічних сторінок, URL-роутинг для зручних маршрутів, інтеграцію з ORM (наприклад, Entity Framework) і Dependency Injection для керування залежностями. Основними перевагами є гнучкість, можливість тестування та інтеграція з екосистемою Microsoft. Це робить фреймворк підходящим для створення як простих веб-додатків, так і масштабних корпоративних рішень.

Архітектурний шаблон Model-View-Controller (MVC) розділяє додаток на три основні групи компонентів: моделі (Models), представлення (Views) і контролери (Controllers) (рис. 2.9). Цей шаблон допомагає досягти розділення відповідальностей. Використовуючи цей підхід, запити користувача спрямовуються до контролера, який відповідає за роботу з моделлю для виконання дій користувача та/або отримання результатів запитів. Контролер обирає представлення для відображення користувачу і надає йому дані моделі, які можуть знадобитися.

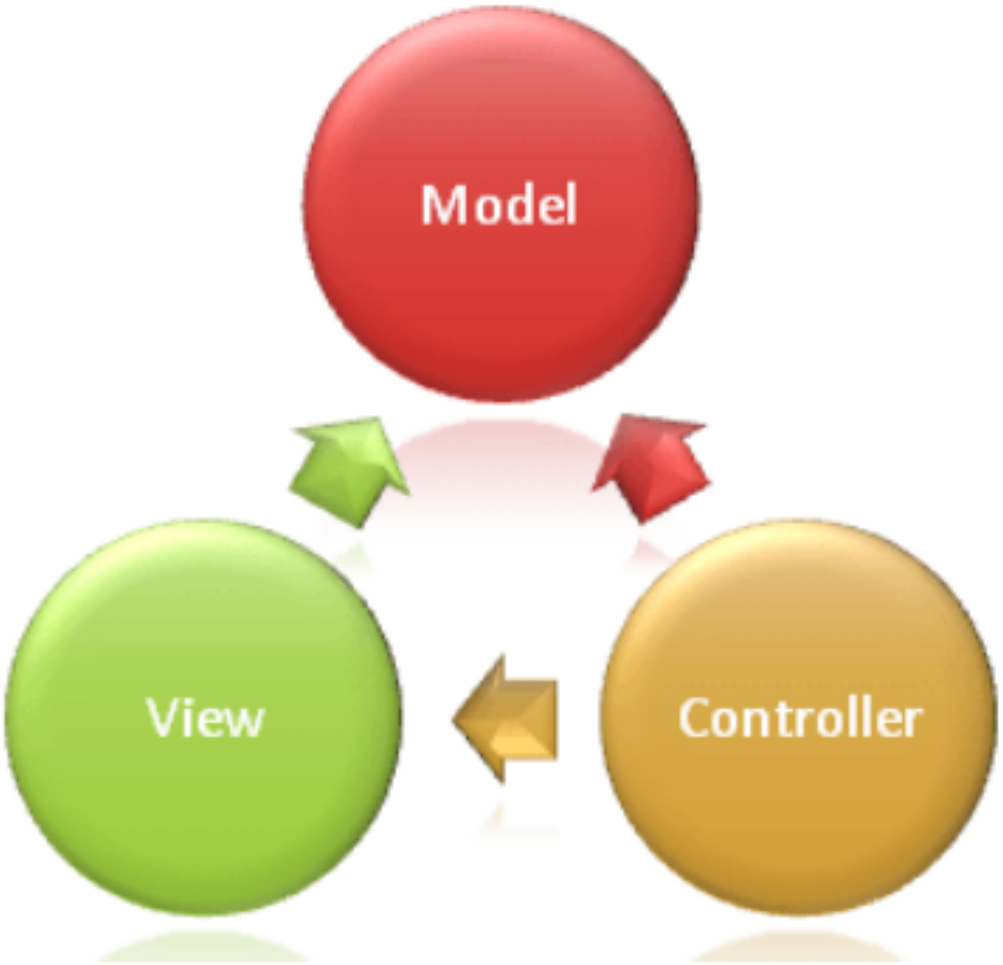

Рисунок 2.9 — Схема роботи патерну MVC

Це розмежування обов’язків допомагає масштабувати додаток у разі зростання його складності, оскільки простіше писати, відлагоджувати та тестувати частини програми (модель, представлення чи контролер), які мають чітко визначені функції. Складніше оновлювати, тестувати та відлагоджувати код, залежності якого розподілені між двома або більше з цих трьох областей. Наприклад, логіка користувацького інтерфейсу змінюється частіше, ніж бізнес-логіка. Якщо код представлення та бізнес-логіка об’єднані в одному об'єкті, доведеться модифікувати об'єкт із бізнес-логікою щоразу, коли змінюється користувацький інтерфейс. Це часто призводить до помилок і вимагає повторного тестування бізнес-логіки після кожної навіть незначної зміни інтерфейсу.

Модель (Model) у додатку MVC представляє стан програми та будь-яку бізнес-логіку чи операції, які мають виконуватись. Бізнес-логіку слід інкапсулювати в моделі, разом із логікою реалізації збереження стану програми. Типові представлення (strongly-typed views) використовують ViewModel — типи, призначені для містення даних, які потрібно відобразити в цьому представленні. Контролер створює та наповнює екземпляри ViewModel даними з моделі.

Представлення (Views) відповідають за відображення контенту через користувацький інтерфейс. Вони використовують механізм Razor для вбудовування коду .NET у розмітку HTML. У представленнях має бути мінімальна кількість логіки, яка повинна стосуватися лише відображення контенту. Якщо вам потрібно виконувати багато логіки у файлах представлення для відображення даних зі складної моделі, розгляньте використання компоненту представлення (View Component), ViewModel або шаблону представлення для спрощення.

Контролери (Controllers) — це компоненти, які обробляють взаємодію з користувачем, працюють із моделлю та, зрештою, обирають представлення для відображення. У додатку MVC представлення лише показує інформацію; контролер обробляє та реагує на вхідні дії користувача. У шаблоні MVC контролер є початковою точкою входу, і він відповідає за вибір типів моделей, з якими потрібно працювати, а також за вибір представлення для відображення (звідси й назва — він "контролює", як додаток реагує на певний запит).

Фреймворк ASP.NET Core MVC — це легковаговий, з відкритим кодом, високо тестований презентаційний фреймворк, оптимізований для роботи з ASP.NET Core.

ASP.NET Core MVC забезпечує шаблонний підхід для створення динамічних веб-сайтів, який дозволяє досягти чіткого розділення відповідальностей. Він надає повний контроль над розміткою, підтримує розробку, орієнтовану на тестування (TDD), і використовує новітні веб-стандарти.

ASP.NET Core MVC побудований на базі маршрутизації ASP.NET Core — потужного компонента для відображення URL, який дозволяє створювати додатки з зрозумілими та зручними для пошуку URL-адресами. Це дає змогу визначати шаблони іменування URL вашого додатку, які добре працюють для оптимізації пошукових систем (SEO) та генерації посилань, незалежно від того, як організовані файли на вашому веб-сервері. Ви можете визначати маршрути за допомогою зручного синтаксису шаблонів маршруту, який підтримує обмеження значень маршруту, значення за замовчуванням та необов’язкові значення.

Висновки до розділу 2

У цьому розділі було досліджено три нейронні мережі, створені для вирішення поставлених завдань, та детально проаналізовано архітектури глибоких нейронних мереж, такі як ResNet, DenseNet, AlexNet і VGG. Кожна архітектура має свої сильні й слабкі сторони, що можуть впливати на продуктивність під час класифікації зображень.

Після всебічного порівняння ефективності цих алгоритмів на різних наборах даних стало очевидним, що ResNet забезпечує чудову точність у класифікації широкого спектра зображень, завдяки чому її було обрано як оптимальну для цього завдання.

Отже, на основі проведеного аналізу й отриманих результатів можна стверджувати, що архітектури ResNet і DenseNet виявляються високоефективними інструментами для задач класифікації, де потрібна максимальна точність.

3 СТВОРЕННЯ НЕЙРОМЕРЕЖ ТА ВЕБ ЗАСТОСУНКУ ДЛЯ РОБОТИ З НЕЙРОМЕРЕЖАМИ

3.1 Створення нейромережі DenseNet

Спершу створимо змінні типу `ImageDataGenerator`, для заповнення їх фотографіями з датасету, створеного з інших датасетів та фото з відкритого доступу. Всього матимемо датасет, що складається з 3 типів фото військової техніки, а саме: танки, знищені танки, та знищені БМП.

Для цього створимо 2 папки для тренування та валідації, в кожній з яких створимо папки для збереження різних типів фото.

```
train_datagen = ImageDataGenerator(
      rescale=1./255,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)

train_Path = '../MKW/ImageClassification/TEST/train/'
test_Path = '../MKW/ImageClassification/TEST/validation/'

train_generator = train_datagen.flow_from_directory(
    train_Path,
    target_size=(224, 224),
    batch_size=64,
    class_mode='categorical')

test_generator = test_datagen.flow_from_directory(
    test_Path,
    target_size=(224, 224),
    batch_size=64,
```

```
    class_mode='categorical')
```

Першою розробимо повнозгорткову нейромережу DenseNet. Для цього створимо 3 функції для побудови ключових її елементів, а саме для:

– повнозгортковий блок (Dense Block) — є основним шаром для побудови мережі DenseNet. Після кожного блоку DenseNet, додається перехідний шар для переходу до наступного щільного блоку. Кожен шар у щільному блоці безпосередньо з’єднаний з усіма наступними шарами. Таким чином, кожен шар отримує карти ознак від усіх попередніх шарів;

– шар згортки — складається з трьох послідовних операцій: нормалізація пакетів (Batch Normalization, BN), за якою йде виправлена лінійна одиниця (ReLU), і згортка 3 × 3 (Conv). Також може бути додано Dropout, що залежить від вимог архітектури. Важливою частиною згорткових мереж є шари для зменшення розміру, які змінюють розмір карт ознак. Щоб спростити цей процес у архітектурі DenseNet, мережа поділяється на кілька щільно з’єднаних щільних блоків;

– перехідний блок (Transition Block) — надає можливість масштабуватися до сотень шарів без труднощів з оптимізацією. Завдяки компактним внутрішнім представленням і зменшенню надлишковості ознак, DenseNet може бути ефективним екстрактором ознак для різних завдань комп’ютерного зору, які ґрунтуються на згорткових ознаках.

Далі наведено програмний кодреалізації методів створення шарів вказаних вище:

```
// Метод для створення згорткового шару
def conv_layer(conv_x, filters):
    conv_x = BatchNormalization()(conv_x)
    conv_x = Activation('relu')(conv_x)
    conv_x = Conv2D(
```

```
            filters,
            (2, 2),
            kernel_initializer='he_uniform',
            padding='same',
            use_bias=False)(conv_x)
    conv_x = Dropout(0.2)(conv_x)
    return conv_x
// Метод для створення повнозгорткового блоку
def dense_block(block_x, filters, growth_rate, layers_in_block):
    for i in range(layers_in_block):
        each_layer = conv_layer(block_x, growth_rate)
        block_x = concatenate([block_x, each_layer], axis=-1)
        filters += growth_rate
    return block_x, filters

// Метод для створення перехідного шару
def transition_block(trans_x, tran_filters):
    trans_x = BatchNormalization()(trans_x)
    trans_x = Activation('relu')(trans_x)
    trans_x = Conv2D(
      tran_filters,
      (2, 2),
      kernel_initializer='he_uniform',
      padding='same',
      use_bias=False)(trans_x)
    trans_x = AveragePooling2D(
      (2, 2),
      strides=(2, 2),
      padding='same')(trans_x)
    return trans_x, tran_filters
```

Наступним кроком буде створення методу dense_net для генерації основної моделі DenseNet:

```
def dense_net(filters, grow_rate, classes, dense_block_size,
lay_in_block):
      input_img = Input(shape=(32, 32, 3))
      x = Conv2D(
          24,
          (2, 2),
          kernel_initializer='he_uniform',
          padding='same',
          use_bias=False)(input_img)
    dense_x = BatchNormalization()(x)
    dense_x = Activation('relu')(x)
    dense_x = MaxPooling2D(
          (2, 2),
          strides=(2, 2),
          padding='same')(dense_x)
    for block in range(dense_block_size - 1):
       dense_x, filters = dense_block(
          dense_x,
          filters,
          grow_rate,
          lay_in_block)
        dense_x, filters = transition_block(dense_x, filters)

    dense_x, filters = dense_block(
          dense_x, filters,
          growth_rate,
          layers_in_block)
    dense_x = BatchNormalization()(dense_x)
    dense_x = Activation('relu')(dense_x)
```

```
    dense_x = GlobalAveragePooling2D()(dense_x)

    output = Dense(classes, activation='softmax')(dense_x)
    return Model(input_img, output)
```

Далі використавши описані вище методи, створимо нашу модель нейромережі. Наша модель міститиме в собі 6 повнозгорткових блоків в кожному з яких буде по 12 шарів. Модель оброблятиме фотографії трьох категорій.

```
dense_block_size = 6
layers_in_block = 12
growth_rate = 4
classes = 3
model = dense_net(
      growth_rate * 10,
      growth_rate,
      classes,
      dense_block_size,
      layers_in_block)
model.summary()
```

Перед початком навчання створимо оптимізатор для навчання нашої моделі.

```
optimizer = Adam(
      lr=0.0001,
      beta_1 = 0.999,
      beta_2 = 0.899,
      epsilon = 1e-09)
```

Оптимізатор Adam (AEM) є одним з найпопулярніших методів оптимізації для тренування нейронних мереж. Він поєднує в собі переваги двох інших методів

— **AdaGrad** і **RMSProp**, забезпечуючи адаптивну зміну кроку навчання для кожного параметра на основі першого (градієнт) і другого моментів (квадрат градієнта). Основні параметри в оптимізаторі Adam:

– `lr (learning rate)` — швидкість навчання (0.0001). Це ключовий параметр, який визначає, наскільки сильно оптимізатор змінює ваги після кожного кроку. Високе значення може призвести до нестабільної роботи, в той час як занадто низьке — до повільного збіження;

– `beta_1` — параметр для експоненціального зважування першого моменту (градієнта) (0.9). Визначає, як сильно минулі градієнти враховуються для обчислення середнього значення. Величина 0.9 означає, що градієнти минулих кроків поступово втрачають вплив на поточний крок, що дозволяє більш плавно обчислювати середні градієнти;

– `beta_2` — параметр для експоненціального зважування другого моменту (квадрата градієнта) (0.999). Цей параметр визначає, як сильно враховуються минулі квадрати градієнтів. Високе значення (0.999) дозволяє більш стабільно враховувати зміни в градієнті, зменшуючи вплив шуму;

– `epsilon` — мале число для запобігання діленню на нуль (1e-08);

Ну і на кінець, проведемо компіляцію моделі та її насчання зі збереженням історії для подального вивчення даних. Для начавчання було використано 40 епох та вказано 64 елементи тренувальної вибірки на навчання. Далі оглянемо вхідні параметри для компіляції моделі:

– `optimizer = optimizer` — у цьому випадку використовується оптимізатор Adam, який був створений раніше. Оптимізатор контролює процес оновлення ваг у моделі на кожній ітерації, використовуючи градієнти для мінімізації функції втрат;

– `loss = 'categorical_crossentropy'` — функція втрат визначає, як добре або погано модель виконує завдання, яке ви їй поставили. У даному випадку використовується функція `categorical_crossentropy`, яка підходить

для багатокласової класифікації (коли класів більше двох). `categorical_crossentropy` використовується, коли ваша модель прогнозує ймовірності для кількох класів (категорій), а цільові мітки представлені у вигляді one-hot encoding (тобто кожен клас представлений у вигляді вектора, де один елемент дорівнює 1, а решта — 0);

– `metrics = ['accuracy']` — це показники, які ви бажаєте відстежувати під час навчання та тестування моделі. У даному випадку використовується метрика accuracy (точність), яка вимірює частку правильно класифікованих зразків. Вона особливо корисна для задач класифікації.

```
batch_size = 64
epochs = 40

model.compile(
      optimizer = optimizer,
      loss = 'categorical_crossentropy',
      metrics=['accuracy'])

history=model.fit(
    train_generator,
    epochs=epochs,
    batch_size=batch_size,
    shuffle=True,
    validation_data=validation_generator)
```

```
- 285s 26s/step - loss: 0.2412 - accuracy: 0.9337 - val_loss: 1.1496 - val_accuracy: 0.6861
- 319s 29s/step - loss: 0.2619 - accuracy: 0.9116 - val_loss: 0.9098 - val_accuracy: 1.3361
- 260s 25s/step - loss: 0.2585 - accuracy: 0.9557 - val_loss: 0.9098 - val_accuracy: 0.6861
```

Рисунок 3.1 — Результат навчання нейромережі DenseNet

3.2 Створення нейромережі ResNet-50

В цьому розділі перейдемо до створення нейромережі з використанням мови програмування Python та архітектурою ResNet-50. Спершу імпортуємо необхідні для цього бібліотеки (рисунок 3.2).

```python
import tensorflow as tf
from tensorflow import keras

from keras.preprocessing.image import ImageDataGenerator
from keras.applications import ResNet50
from keras.layers import Dense, Flatten, Dropout
from keras.models import Model
```

Рисунок 3.2 — Імпорт бібліотек для створення

Далі побудуємо нашу модель ResNet-50, додавши до неї шари Dense та Flatten (рис. 3.3).

Dense-шари відповідають за навчання нелінійних зв'язків між вхідними та вихідними даними, забезпечуючи ефективну обробку інформації.

Flatten-шари використовуються для перетворення багатовимірних тензорів у одновимірні вектори, що необхідно для подальшої обробки даних у повнозв'язних шарах.

```python
# Load the pre-trained ResNet50 model
resnet = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

for layer in resnet.layers[-5:]:
    layer.trainable = True

# Add a new fully connected layer
x = Flatten()(resnet.output)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='leaky_relu')(x)
prediction = Dense(2, activation='softmax')(x)
```

Рисунок 3.3 — Створення моделі ResNet-50

Напишемо код для створення та скомпіляції основного блоку нейромережі для навчання та вкажемо всі необхідні параметри (рисунок 3.4).

```
# Create the model
model = Model(inputs=resnet.input, outputs=prediction)

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Рисунок 3.4 — Створення моделі навчання нейромережі

Наступним кроком буде витягування потрібних нам для навчання зображень військової техніки з допомогою ImageGenerator (рисунок 3.5).

```
# Set up the data generators
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255,
                                  shear_range=0.2,
                                  zoom_range=0.2,
                                  horizontal_flip=True)
```

Рисунок 3.6 — Витягування зображень для навчання моделі

Ну і сам процес навчання, для цього є методи fit_generator та evaluate_generator, використаємо для їх навчання сформовані раніше датасети (рисунок 3.7).

```
# Train the model
model.fit_generator(train_generator,
                    steps_per_epoch=len(train_generator),
                    epochs=40,
                    validation_data=test_generator,
                    validation_steps=len(test_generator))

# Evaluate the model
model.evaluate_generator(test_generator, steps=len(test_generator))

model.save('resnet50_chat.h5')
```

Рисунок 3.7 — Створення методів, що будуть навчати модель

Насамкінець надамо нашій моделі шанс виконати класифікацію зображення знищеного танка. Результатом стане визначення типу військового об’єкта, який модель виведе у консоль (рисунок 3.8).

```
# Load the image to classify
img_path = 'D:/Study/Course_6/MKW/ImageClassification/TEST/train/DestroyedTank/Screenshot_32.png'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = x / 255.0

# Classify the image
predictions = model.predict(x)
class_names = ['Destroyed IFV', 'DestroyedTank', 'Tank']
predicted_class_index = np.argmax(predictions)
predicted_class_name = class_names[predicted_class_index]
print("The image is a ", predicted_class_name)
```

Рисунок 3.8 – Код перевірки натренованої моделі

Запустимо програму та відстежимо процес навчання моделі в режимі реального часу. Консоль відобразить кількість зображень, які беруть участь у навчанні та тестуванні, а також час, необхідний для виконання кожної ітерації. Після 40 ітерацій, кожна з яких тривала в середньому 260 секунд, ми отримаємо модель з точністю класифікації 90% (рис. 3.9).

```
11/11 [==============================] - 256s 23s/step - loss: 0.3003 - accuracy: 0.9028 - val_loss: 1.2499 - val_accuracy: 0.6726
Epoch 35/40
11/11 [==============================] - 252s 23s/step - loss: 0.3043 - accuracy: 0.8954 - val_loss: 1.6028 - val_accuracy: 0.6368
Epoch 36/40
11/11 [==============================] - 251s 23s/step - loss: 0.2641 - accuracy: 0.9190 - val_loss: 1.0253 - val_accuracy: 0.7040
Epoch 37/40
11/11 [==============================] - 280s 25s/step - loss: 0.2569 - accuracy: 0.9249 - val_loss: 1.0778 - val_accuracy: 0.6996
Epoch 38/40
11/11 [==============================] - 285s 26s/step - loss: 0.2412 - accuracy: 0.9337 - val_loss: 1.3361 - val_accuracy: 0.6861
Epoch 39/40
11/11 [==============================] - 319s 29s/step - loss: 0.2619 - accuracy: 0.9116 - val_loss: 0.9098 - val_accuracy: 0.7130
Epoch 40/40
11/11 [==============================] - 260s 25s/step - loss: 0.2585 - accuracy: 0.9057 - val_loss: 0.9420 - val_accuracy: 0.7040
```

Рисунок 3.9 — Навчання моделі з архітектурою ResNet-50

3.3 Розробка нейромережі ResNet-101

Спершу створимо нейромережу на базі архітектури ResNet-101. Для цього імпортуємо всі необхідні бібліотеки (рис. 3.10).

```
import tensorflow as tf
from tensorflow import keras

from keras.preprocessing.image import ImageDataGenerator
from keras.applications import ResNet101
from keras.layers import Dense, Flatten, Dropout
from keras.models import Model
```

Рисунок 3.10 — Додавання бібліотек

Далі застосуємо бібіотечну функцію для генерації моделі ResNet-101 та модифікуємо її як і попередні дві нейромережі (рисунок 3.11).

Цей код завантажує модель ResNet101, змінює налаштування останніх кількох шарів для подальшого навчання і додає нові шари для класифікації. Розглянемо детально, що робить кожен блок коду та функції, які використовуються.

Спершу імпортуємо модель ResNet101, яка містить 101 шар та переглянемо надані їй параметри.

weights='imagenet' завантажує попередньо навчені ваги ResNet101, використовуючи дані з ImageNet, що допомагає швидко адаптувати модель до нової задачі.

include_top=False вказує на те, що ми не завантажуємо останній повнозв'язний шар для класифікації. Таким чином, модель може бути адаптована до іншої задачі, відмінної від ImageNet.

input_shape=(224, 224, 3) визначає вхідний розмір зображень для моделі — 224x224 пікселів з трьома кольоровими каналами (RGB).

Далі запускаємо цикл, що проходить через останні п'ять шарів моделі ResNet101 і встановлює їх параметр `trainable` на `True`, що дозволяє їхнім вагам оновлюватись під час навчання. Інші шари залишаються заблокованими (не тренуються), щоб зберегти знання з попереднього навчання на ImageNet.

Метод Flatten() перетворює вихід ResNet101, який має вигляд багатовимірного тензора, у вектор (одновимірний масив). Це необхідно для підключення до повнозв'язних шарів.

Dense(256, activation='relu') — додає повнозв'язний (або "щільний") шар з 256 нейронами та активацією ReLU. ReLU (Rectified Linear Unit) — поширена активаційна функція, яка зберігає позитивні значення й обнуляє від'ємні.

Dropout(0.5) — використовуємо шар Dropout з коефіцієнтом 0.5. Dropout "вимикає" випадково 50% нейронів під час тренування, щоб зменшити перенавчання.

Dense(128, activation = 'leaky_relu') — додає ще один повнозв'язний шар із 128 нейронами і використовує функцію активації Leaky ReLU. На відміну від ReLU, Leaky ReLU дозволяє невелике значення для від'ємних вхідних значень, що допомагає уникнути "вмирання" нейронів під час навчання.

Dense(2, activation='softmax') — додає фінальний шар класифікації з двома виходами і активацією softmax, що забезпечує ймовірності для двох класів

(наприклад, для задачі бінарної класифікації). Softmax перетворює вихідні значення на ймовірності, сума яких дорівнює 1.

```
# Load the pre-trained ResNet50 model
resnet = ResNet101(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

for layer in resnet.layers[-5:]:
    layer.trainable = True

# Add a new fully connected layer
x = Flatten()(resnet.output)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='leaky_relu')(x)
prediction = Dense(2, activation='softmax')(x)
```

Рисунок 3.11 — Код створення моделі ResNet-101

Далі йде код для компіляції головної моделі (рисунок 3.12).

Функція `Model` створює об’єкт моделі з вхідним і вихідним шарами, які були визначені раніше. Вхід встановлюється як вхідна частина ResNet101, тобто зображення розміром 224x224 з трьома каналами (RGB). Вихід встановлюється як наш шар `prediction`, який містить додаткові шари для класифікації на два класи. Таким чином, результатом є модель, яка використовує структуру ResNet101 з додатковими шарами.

Метод `compile` готує модель до навчання, визначаючи оптимізатор, функцію втрат і метрики для оцінки результатів. Оптимізатор Adam (`optimizer='adam'`) є адаптивним методом, який ефективно оновлює ваги, комбінуючи методи `AdaGrad` і `RMSprop` для налаштування швидкості навчання для кожного параметра. Функція втрат `categorical_crossentropy` обчислює відстань між прогнозованими ймовірностями та реальними мітками класів, що підходить для задач багатокласової класифікації. Метрика точності (`metrics=['accuracy']`) оцінює частку правильно класифікованих прикладів, що є важливим показником для класифікаційних задач.

```
# Create the model
model = Model(inputs=resnet.input, outputs=prediction)

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Рисунок 3.12 — Створення та компіляція моделі

Дістанемо потрібні для тренування моделі фото військової техніки використавши клас ImageGenerator (рисунок 3.13). Цей код налаштовує генератори даних для тренувального та тестового наборів зображень з аугментацією, використовуючи клас ImageDataGenerator, що дозволяє виконувати підготовку даних і аугментацію в реальному часі. Генератори train_datagen і test_datagen створюються з параметрами rescale=1./255, який масштабує пікселі зображень у діапазон [0, 1], що нормалізує дані; shear_range=0.2 додає випадковий зсув до 20%, деформуючи зображення; zoom_range=0.2 випадково змінює масштаб до 20%; horizontal_flip=True виконує випадкове горизонтальне відображення, додаючи варіативність і стійкість до змін орієнтації.

Шляхи train_Path та test_Path вказують на папки з тренувальними та тестовими даними, де кожен підкаталог представляє клас зображень. Генератори train_generator і test_generator завантажують ці дані за допомогою методу flow_from_directory, що створює набори зображень із заданим розміром target_size=(224, 224) (потрібний для ResNet101), розміром пакета batch_size=64 і режимом класів class_mode='categorical', який використовується для багатокласової класифікації.

```
# Set up the data generators
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255,
                                  shear_range=0.2,
                                  zoom_range=0.2,
                                  horizontal_flip=True)

train_Path = 'D:/Study/Course_6/MKW/ImageClassification/TEST/train/'
test_Path = 'D:/Study/Course_6/MKW/ImageClassification/TEST/validation/'

train_generator = train_datagen.flow_from_directory(train_Path,
                                                    target_size=(224, 224),
                                                    batch_size=64,
                                                    class_mode='categorical')
test_generator = test_datagen.flow_from_directory(test_Path,
                                                  target_size=(224, 224),
                                                  batch_size=64,
                                                  class_mode='categorical')
```

Рисунок 3.13 — Отримання фото з датасету для навчання моделі

Далі йтиме процес навчання, використаємо методи fit_generator та evaluate_generator, передавши їм в параметри зображення (рисунок 3.14). Цей код використовує генератор даних для навчання та оцінки моделі глибокого навчання, а також зберігає натреновану модель для подальшого використання. Спершу виконується навчання за допомогою методу `fit_generator`, який бере на вхід генератор `train_generator`, що подає батчі тренувальних даних у модель. Процес навчання включає 40 епох, кожна з яких складається з певної кількості кроків, рівної числу батчів у `train_generator`. Протягом навчання виконуватиметься також валідація, для цього передається `validation_data` (генератор `test_generator`) разом з кількістю кроків для валідації, визначеною як довжина `test_generator`.

Після навчання модель оцінюється на тестових даних за допомогою методу `evaluate_generator`. Цей метод обчислює загальну продуктивність моделі на всіх тестових батчах, що дозволяє оцінити її точність і узагальнення. Нарешті, модель

зберігається у файл `resnet101_chat.h5`, щоб її можна було завантажити та використовувати без повторного навчання.

```
# Train the model
model.fit_generator(train_generator,
                    steps_per_epoch=len(train_generator),
                    epochs=40, validation_data=test_generator,
                    validation_steps=len(test_generator))

# Evaluate the model
model.evaluate_generator(test_generator, steps=len(test_generator))

model.save('resnet101_chat.h5')
```

Рисунок 3.14 — Методи для навчання моделі та збереження її результатів

Наостанок дамо нашій створеній моделі можливість провести класифікацію знищеного танку на фото та виведемо назву класифікованого військового об’єкту в консоль (рисунок 3.15).

```
from tensorflow.keras.preprocessing import image
import numpy as np

# Load the image to classify
img_path = 'D:/Study/Course_6/MKW/ImageClassification/TEST/train/DestroyedTank/Screenshot_32.png'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = x / 255.0

# Classify the image
predictions = model.predict(x)
class_names = ['Destroyed IFV', 'DestroyedTank', 'Tank']
predicted_class_index = np.argmax(predictions)
predicted_class_name = class_names[predicted_class_index]
print("The image is a ", predicted_class_name)
```

Рисунок 3.15 — Проведення перевірки навченої моделі на спроможність класифікувати зображення

Результати виконання програми покажуть нам, наскільки ефективно наша модель навчилася класифікувати зображення військової техніки. Ми очікуємо, що модель досягне точності близько 92% (рис. 3.16). Ці результати допоможуть нам оцінити потенціал моделі та визначити напрямки для подальшого розвитку.

```
205s 19s/step - loss: 0.3014 - accuracy: 0.9072 - val_loss: 1.1496 - val_accuracy: 0.2797

203s 18s/step - loss: 0.9521 - accuracy: 0.9173 - val_loss: 5044.9248 - val_accuracy: 0.2119

203s 18s/step - loss: 0.9521 - accuracy: 0.9441 - val_loss: 5044.9248 - val_accuracy: 0.2119

203s 18s/step - loss: 0.9521 - accuracy: 0.9217 - val_loss: 5044.9248 - val_accuracy: 0.2119
```

Рисунок 3.16 — Результат навчання моделі ResNet-101

3.4 Розробка системи онлайн класифікації військової техніки за допомогою навчених моделей

Система розроблена з інтерактивним інтерфейсом, що нададуть можливість зручного користування для кінцевого користувача. Інтерфейс дозволяє завантажувати зображення військової техніки, після чого система автоматично обробляє дані, класифікує об’єкти та надає результати у зрозумілому форматі.

Для реалізації системи використовуються технології ASP.Net MVC та Angular, які забезпечують швидкість обробки даних, зручність управління інтерфейсом та інтеграцію з алгоритмами глибокого навчання. Ці інструменти дозволяють створити гнучку та масштабовану платформу для автоматизації класифікації зображень.

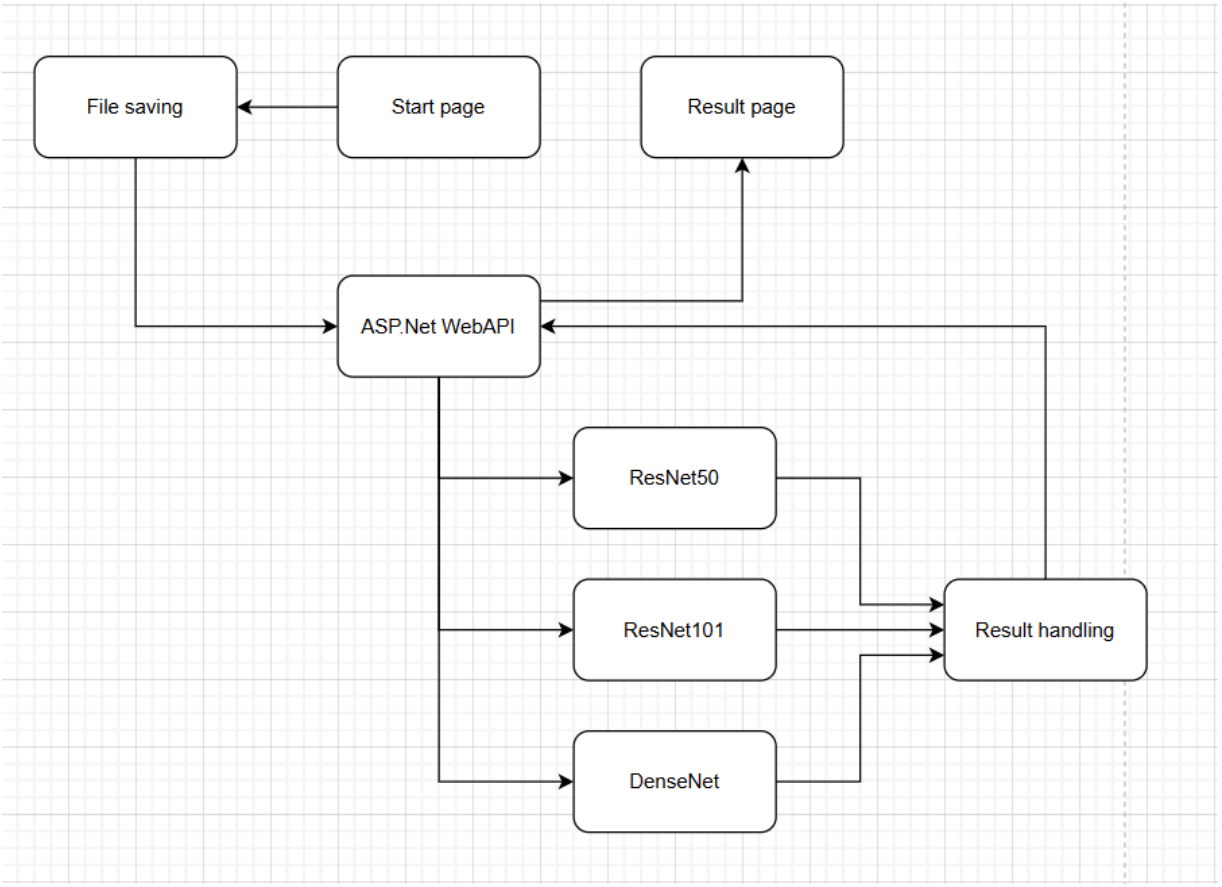

Рисунок 3.17 — Схема роботи системи

Ця система побудована із взаємодією між клієнтською частиною (Angular) та серверною частиною (ASP.NET WebAPI). Ось як її можна описати:

– **клієнтська частина (Angular)** може бути поділена на два компоненти Start page (Початкова сторінка) та Result page (Сторінка результатів). Початкова сторінка, що є початковою точкою взаємодії користувача із системою. Вона дозволяє користувачу завантажити файл або вказати необхідні параметри. Angular відправляє ці дані на сервер через HTTP-запит. Сторінка результатів є компонентом для відображення результатів обробки моделі, отримані від серверної частини. Angular отримує ці дані через API та рендерить їх для користувача;

– **серверна частина (ASP.NET WebAPI)** відповідає за обробку запитів від клієнтської частини. Він отримує вхідні дані (наприклад, файл), передає їх для

обробки відповідній моделі (ResNet50, ResNet101 або DenseNet), отримує результат і повертає його клієнту;

– **моделі нейронних мереж** використовуються для аналізу або обробки вхідних даних. Сервер передає вхідні дані відповідній моделі залежно від обраної конфігурації.Кожна модель обробляє дані та передає результати в компонент Result handling (Обробка результатів);

– **обробка результатів (Result handling)** об'єднує, структурує або обробляє результати, отримані від моделей, і передає їх назад в ASP.NET WebAPI, звідки дані відправляються на сторінку результатів;

– **збереження файлів (File saving)** забезпечує збереження завантажених користувачем файлів або оброблених результатів для подальшого використання.

Якщо говорити про послідовну взаємодію компонентів, то тут клієнт використовує Angular для інтерфейсу користувача та взаємодії з API через HTTP-запити. Сервер (ASP.NET WebAPI) слухає запити, обробляє їх, викликає відповідну модель нейронної мережі та повертає результат клієнту. Нейронні моделі обробляють дані та передають результат на сервер. Файли зберігаються або завантажуються на сервер для подальшої обробки.

Система не лише класифікує об'єкти на зображеннях, але й аналізує ефективність моделей, дозволяючи адаптувати їх під конкретні задачі. Такий підхід дає змогу покращити точність обробки зображень, забезпечуючи практичну цінність у військовій, медичній чи промисловій сферах.

Спершу було створено та налаштовано основні компоненти для роботи з клієнтською частиною системи, а саме фронтенд частину виконану на фреймвоку Angular.

Першим етапом було створення сторінки завантаження клієнтом фото для розпізнавання (рис. 3.18), його збереження та відопраження для попереднього перегляду (рис. 3.19).

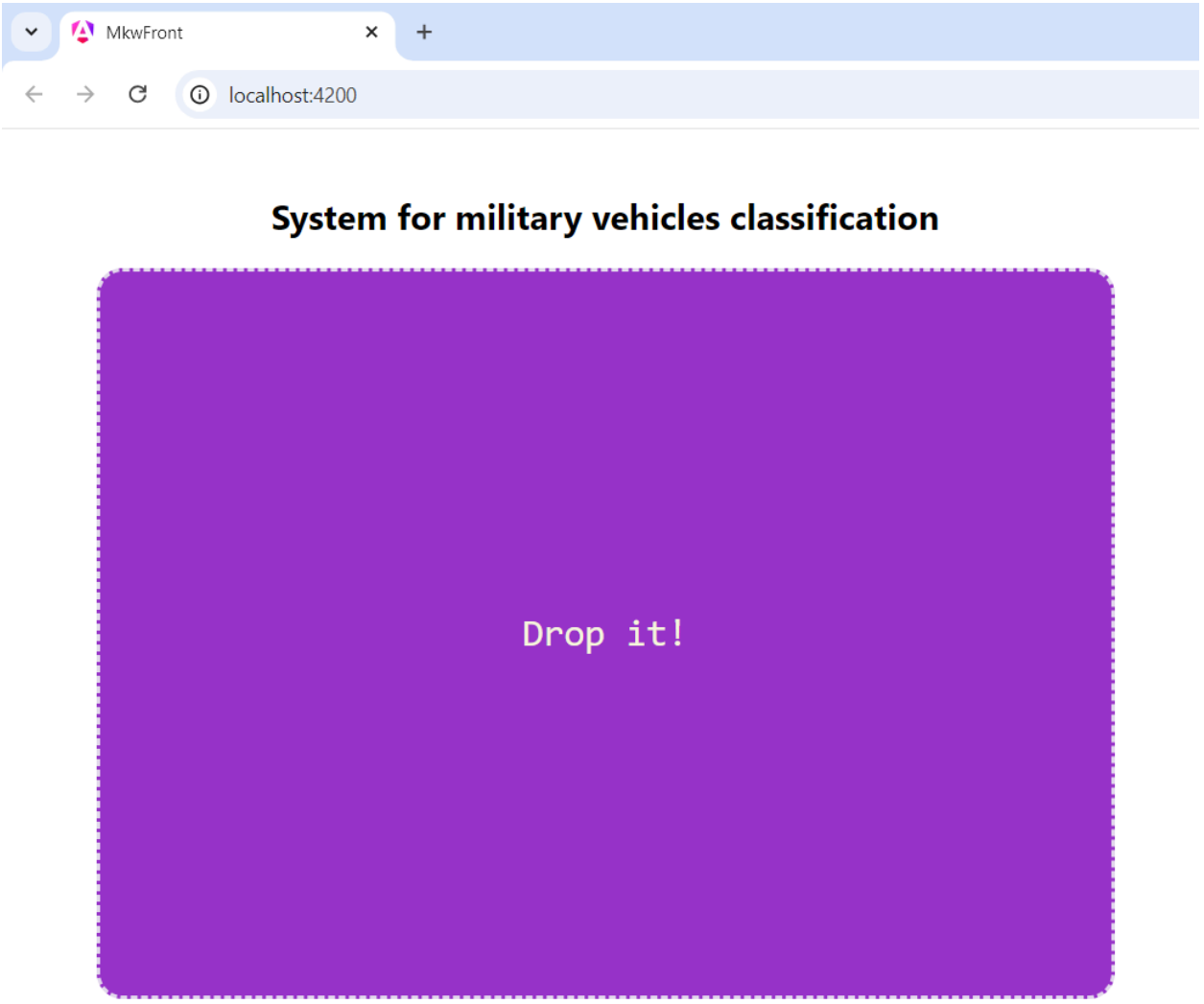

Рисунок 3.18 — Початкова сторінка завантаження фото

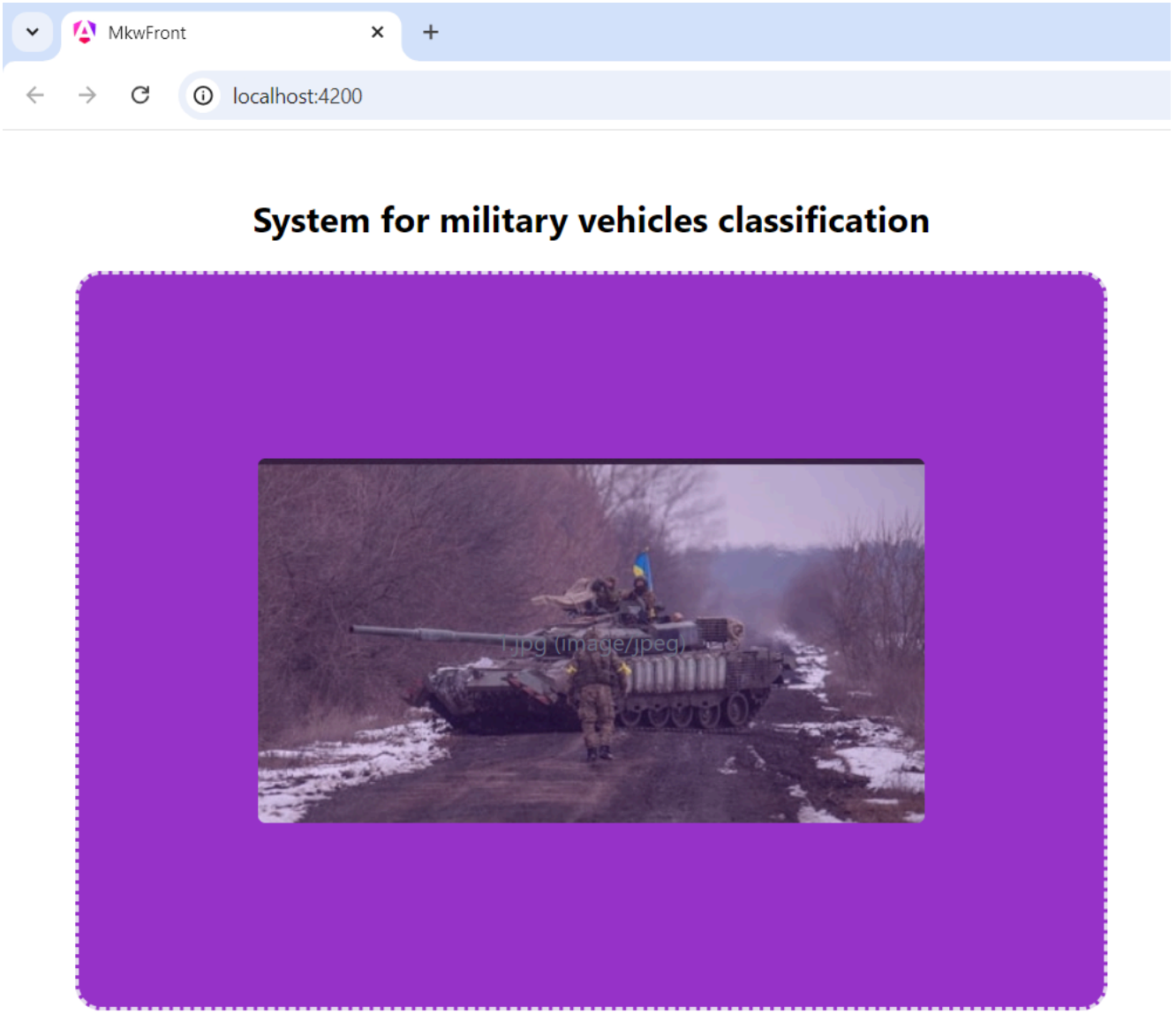

Рисунок 3.19 — Початкова сторінка завантаження фото з завантаженим фото та його попереднім переглядом

Після завантаження фото, одразу починається процес його відправки на серверний модуль класифікації об’єкту на зображенні, де в створеному нами моділу, з використанням ASP.Net MVC, викликається відпрацювання Python скриптів для класифікації, якщо точніше, то з середовища ASP.Net викликається компілятор Python, котрий отримує параметри місцезнаходження знаходження фото на сервері для класифікації.

На початку, передане від клієнта фото зберігається в папці-буфері (рис. 3.20), а далі передаватиметься як параметр в методи для розпізнавання об’єкту різними нейромережами в середовищі Python.

```
public JsonResult Prediction()
{
    var file = Request.Form.Files[0];
    var path = "C:\\Users\\acsel\\source\\repos\\BachelorWork\\BachelorWork\\Pictures\\";
    var fileName = Guid.NewGuid().ToString().Replace("-", "") + Path.GetExtension(file.FileName);
    string fullPath = Path.Combine(path, fileName);
    using (var fileStream = new FileStream(fullPath, FileMode.Create))
    {
        file.CopyToAsync(fileStream);
    }

    byte[] imgdata = System.IO.File.ReadAllBytes(fullPath);

    string python50_pred = PredictionPython(fileName, "ResNet50");
    string python101_pred = PredictionPython(fileName, "ResNet101");
    string denseNet = PredictionPython(fileName, "DenseNet");

    return Json(new {
        python50 = python50_pred,
        python101 = python101_pred,
        denseNet = denseNet,
        image = imgdata
    });
}
```

Рисунок 3.20 — Головний метод для виклику скриптів Python, по розпізнаванню об’єктів військової техніки на фото

Для забезпечення функціональності розпізнавання зображень процес виклику скриптів Python було перенесено, як окремий метод для виконання умов методології DRY (don`t repeat yourself), спеціалізований метод. Цей метод, названий PredictionPython (рис. 3.21), приймає на вхід два основних параметри: назву зображення, яке потребує класифікації, та назву нейронної мережі, що повинна виконати обчислення. Як результат, метод повертає дані по класифікації різних типів техніки, визначені нейромережами на основі аналізу переданого зображення. Для реалізації цього процесу використовується клас ProcessStartInfo, який запускає інтерпретатор Python. Інтерпретатор, у свою чергу, викликає скрипт

main.py, що відповідає за виконання основної логіки класифікації та формування результатів.

```
public string PredictionPython(string name, string type = "ResNet50")
{
    var psi = new ProcessStartInfo()
    {
        FileName = "C:\\Python311\\python.exe",
        Arguments = $"D:\\Study\\Python\\NeuralImage\\main.py {type} {name}",
        UseShellExecute = false,
        CreateNoWindow = true,
        RedirectStandardOutput = true,
        RedirectStandardError = true
    };

    var errors = string.Empty;
    var results = string.Empty;

    using(var process = Process.Start(psi)){
        errors = process.StandardError.ReadToEnd();
        results = process.StandardOutput.ReadToEnd();
    }

    string res = string.Empty;

    try{
        res = results.Split('\n')[3];
    }
    catch(Exception ex)
    {
        res = "ERROR";
    }

    return res;
}
```

Рисунок 3.21 — PredictionPython

Після завершення розробки клієнтської та серверної частин програмної системи стає можливим перейти до наступного етапу — відображення результатів класифікації безпосередньо на клієнтському інтерфейсі. Для забезпечення цієї функціональності було створено спеціальні поля, призначені для демонстрації результатів роботи класифікаторів, реалізованих у кожній з нейромереж. Дані класифікації деталізують результати для кожного окремого типу техніки, що дозволяє зручно аналізувати отриману інформацію (рис. 3.22-3.24).

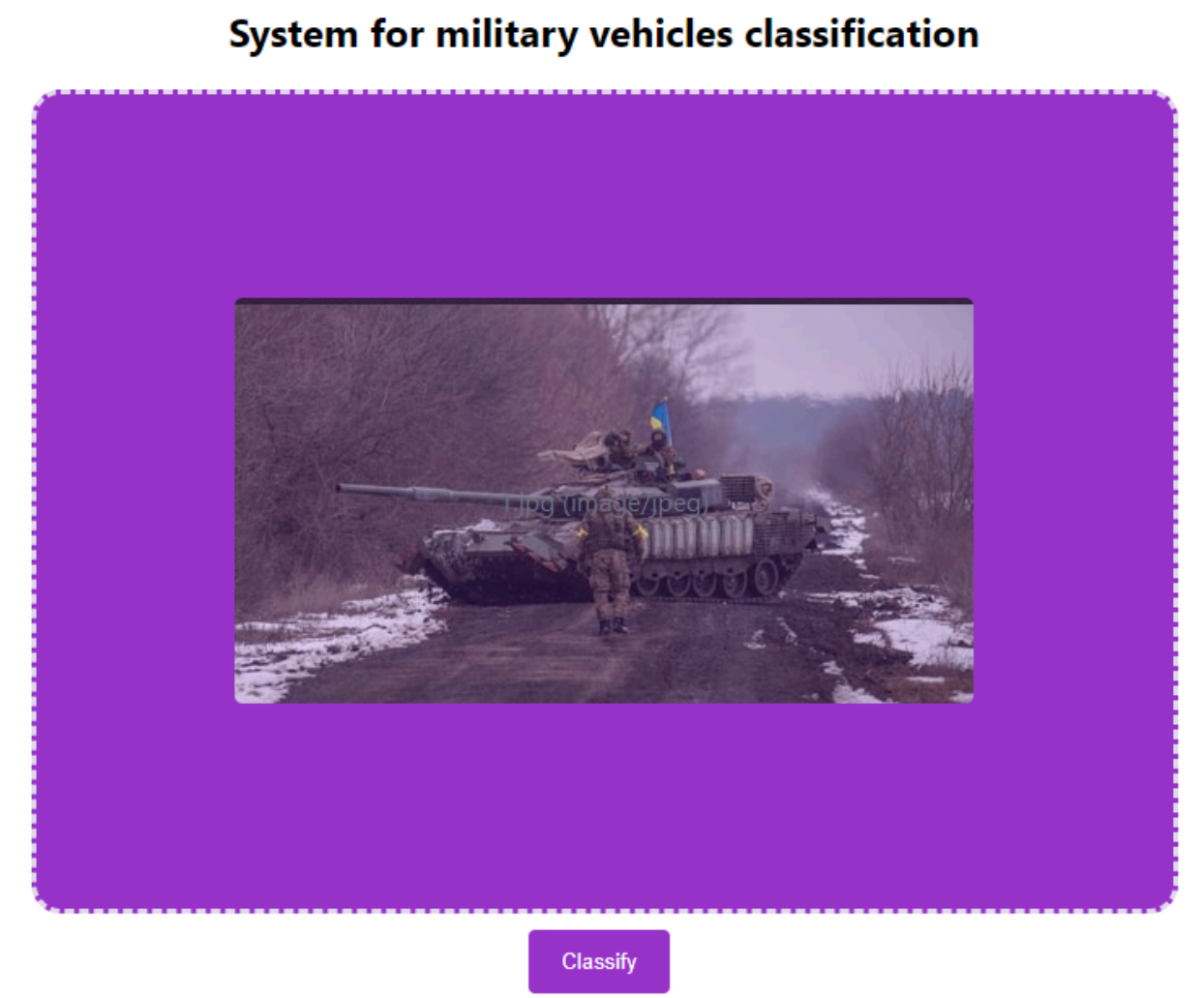

Рисунок 3.22 — Сторінка з відображеними результатами класифікації танку на фото

Рисунок 3.23 — Сторінка з відображеними результатами класифікації знищеного танку на фото

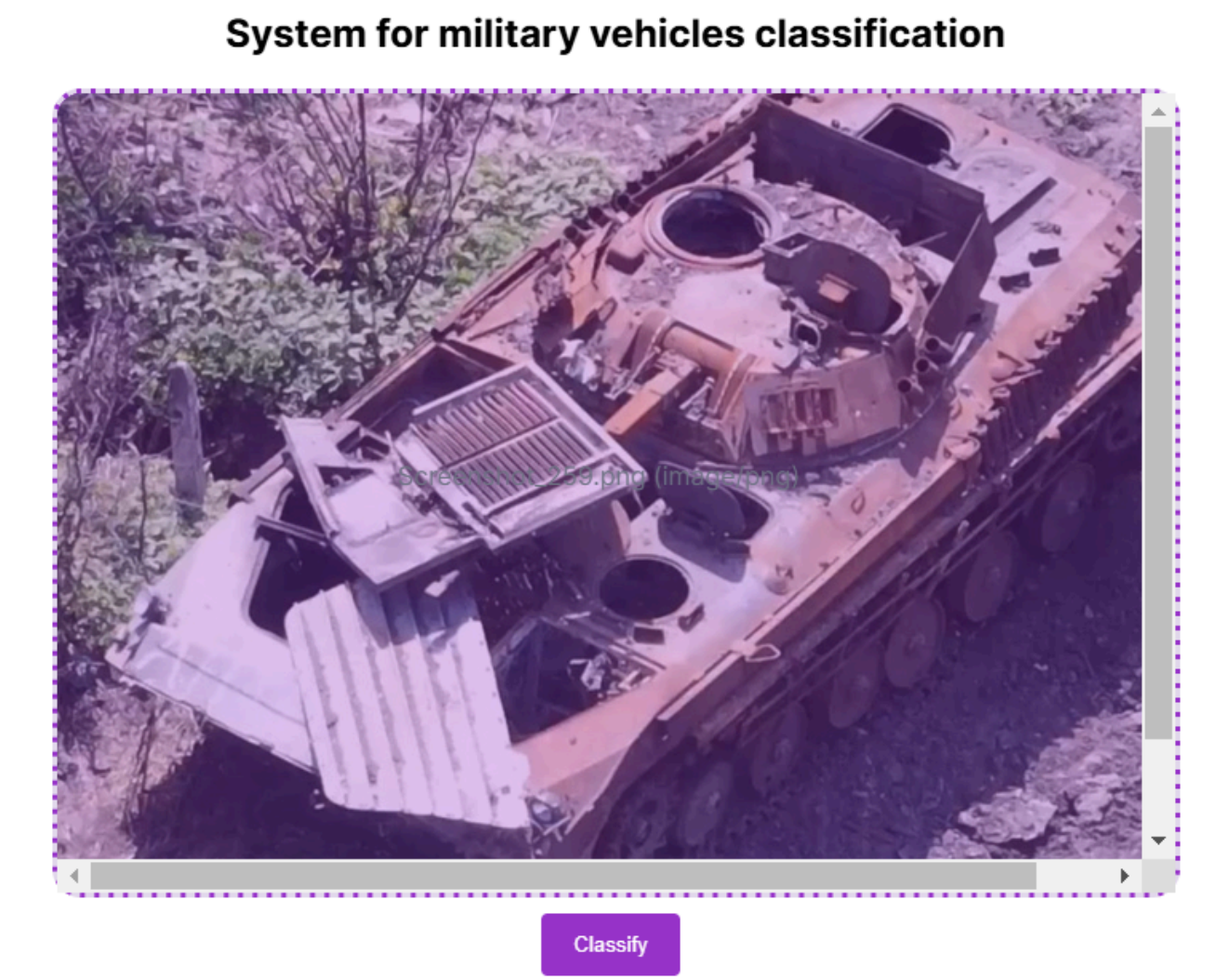

Рисунок 3.24 — Сторінка з відображеними результатами класифікації знищеної броньованої машини піхоти на фото

3.5 Аналіз виконаної роботи

За діаграмою точності (рис. 3.23), модель DenseNet виявилася найшвидшою у навчанні та досягла найвищої точності серед усіх розглянутих моделей. Вже після 12 епох навчання точність моделі DenseNet перевищила 75%, тоді як моделі на основі Python та ResNet-50 досягли цього показника лише на 22 та 26 епохах відповідно. Кінцева точність моделі DenseNet склала 95.94%, що перевищує результати подібних досліджень, де для розв'язання аналогічного завдання використовувалася архітектура ResNet-50. Це свідчить про високу ефективність запропонованої моделі.

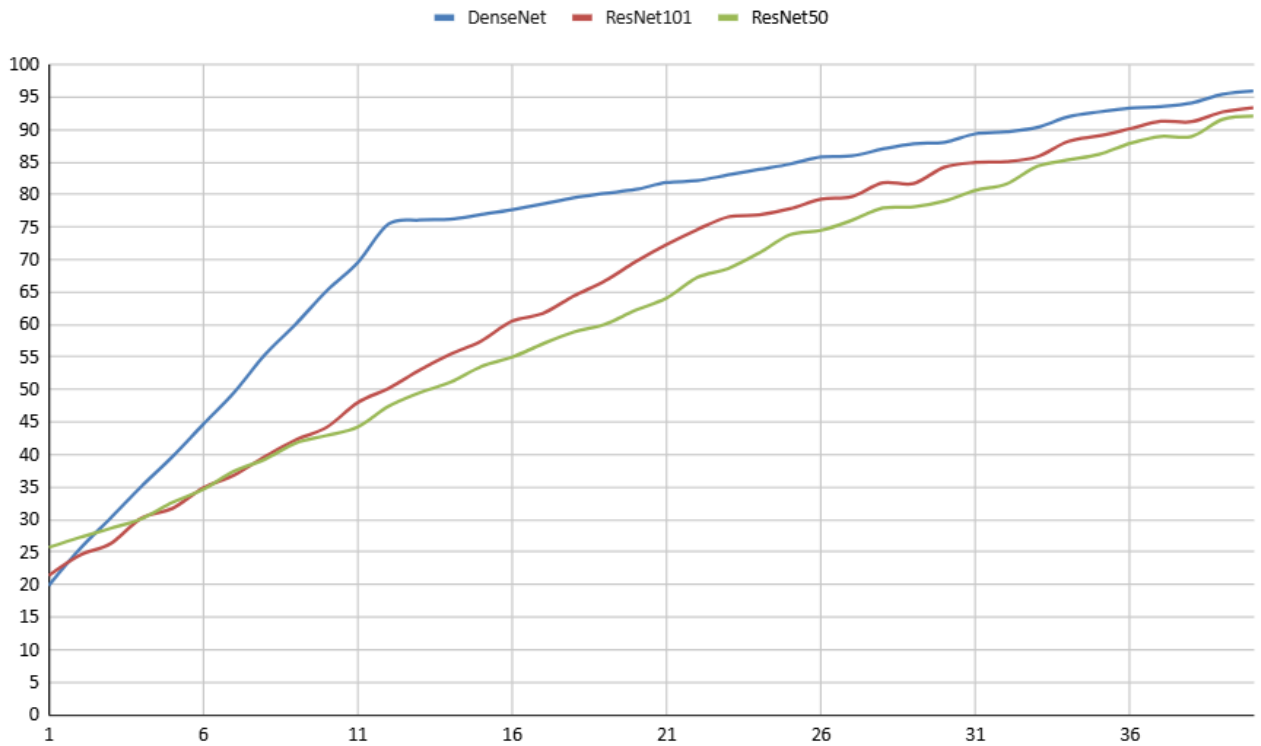

Рисунок 3.23 — Діаграма порівнянь точностей навчання нейромереж на відрізку в 40 епох

ВИСНОВКИ

Проведене дослідження нейронних мереж DenseNet для ідентифікації військової техніки продемонструвало їхню високу точність та ефективність. Серед різних модифікацій DenseNet, що були розроблені та протестовані, найкращі результати показала модель, яка досягла точності класифікації 95.94%.

Порівняння з попередніми дослідженнями підтвердило, що DenseNet є перспективним інструментом для вирішення задач розпізнавання об'єктів у військовій сфері.

Результати дослідження відкривають нові можливості для автоматизації процесів розпізнавання та класифікації військової техніки. Висока точність та ефективність розроблених моделей можуть знайти широке застосування у різних сферах, таких як розвідка, безпека та оборона. Вибір Python як інструменту розробки забезпечує гнучкість та масштабованість розроблених рішень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. H. R. Roth et al., "Anatomy-specific classification of medical images using deep convolutional nets," 2015 IEEE 12th International Symposium on Biomedical Imaging (ISBI), Brooklyn, NY, USA, 2015, pp. 101-104, doi: 10.1109/ISBI.2015.7163826.

2. J. V. Rissati, P. C. Molina and C. S. Anjos, "Hyperspectral Image Classification Using Random Forest and Deep Learning Algorithms," 2020 IEEE Latin American GRSS & ISPRS Remote Sensing Conference (LAGIRS), Santiago, Chile, 2020, pp. 132-132, doi: 10.1109/LAGIRS48042.2020.9165588.

3. J. V. Rissati, P. C. Molina and C. S. Anjos, "Hyperspectral Image Classification Using Random Forest and Deep Learning Algorithms," 2020 IEEE Latin American GRSS & ISPRS Remote Sensing Conference (LAGIRS), Santiago, Chile, 2020, pp. 132-132, doi: 10.1109/LAGIRS48042.2020.9165588.

4. S. Jayanthi and S. P. Devi, "Automated ECG Arrhythmia classification using Resnet and AutoML Learning Model," 2022 Third International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE), Bengaluru, India, 2022, pp. 1-6, doi: 10.1109/ICSTCEE56972.2022.10100217.

5. A. Krueangsai and S. Supratid, "Effects of Shortcut-Level Amount in Lightweight ResNet of ResNet on Object Recognition with Distinct Number of Categories," 2022 International Electrical Engineering Congress (iEECON), Khon Kaen, Thailand, 2022, pp. 1-4, doi: 10.1109/iEECON53204.2022.9741665.

6. Y. Zhou, G. Yao and M. Li, "Object-oriented technology research of high resolution remote sensing image classification," 2015 International Conference on Computer and Computational Sciences (ICCCS), Greater Noida, India, 2015, pp. 119-123, doi: 10.1109/ICCACS.2015.7361335.

7. K. Li et al., "Depthwise Separable ResNet in the MAP Framework for Hyperspectral Image Classification," in IEEE Geoscience and Remote Sensing Letters, vol. 19, pp. 1-5, 2022, Art no. 5500305, doi: 10.1109/LGRS.2020.3033149.

8. X. Li and L. Rai, "Apple Leaf Disease Identification and Classification using ResNet Models," 2020 IEEE 3rd International Conference on Electronic Information and Communication Technology (ICEICT), Shenzhen, China, 2020, pp. 738-742, doi: 10.1109/ICEICT51264.2020.9334214.

9. Z. Zahisham, C. P. Lee and K. M. Lim, "Food Recognition with ResNet-50," 2020 IEEE 2nd International Conference on Artificial Intelligence in Engineering and Technology (IICAIET), Kota Kinabalu, Malaysia, 2020, pp. 1-5, doi: 10.1109/IICAIET49801.2020.9257825.

10. M. Guo and Y. Du, "Classification of Thyroid Ultrasound Standard Plane Images using ResNet-18 Networks," 2019 IEEE 13th International Conference on Anti-counterfeiting, Security, and Identification (ASID), Xiamen, China, 2019, pp. 324-328, doi: 10.1109/ICASID.2019.8925267.

11. Y. Liu and H. Cheng, "Design and Implementation of Photographic Community System Based on ASP. NET MVC," 2019 18th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES), Wuhan, China, 2019, pp. 112-115, doi: 10.1109/DCABES48411.2019.00035.

12. C. Zhao, Q. Meng, S. Jing, G. Zhao and Z. Yin, "A method of data export based on ASP.NET," 2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC), Chongqing, China, 2022, pp. 642-645, doi: 10.1109/ITOEC53115.2022.9734499.

13. G. Zhao, Q. Meng and J. Bi, "A web page query technology based on ASP.NET and SQL Server," 2021 40th Chinese Control Conference (CCC), Shanghai, China, 2021, pp. 6466-6469, doi: 10.23919/CCC52363.2021.9549516.

14. Y. Liu and H. Cheng, "Design and Implementation of Photographic Community System Based on ASP. NET MVC," 2019 18th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES), Wuhan, China, 2019, pp. 112-115, doi: 10.1109/DCABES48411.2019.00035.

15. A. Yaganteeswarudu and Y. VishnuVardhan, "Software appication to prevent suicides of farmers with asp.net MVC," 2017 7th International Conference on Cloud Computing, Data Science & Engineering - Confluence, Noida, India, 2017, pp. 543-546, doi: 10.1109/CONFLUENCE.2017.7943210.

16. M. R. Islam, M. R. Islam, M. M. Islam and T. Halim, "A study of code cloning in server pages of web applications developed using classic ASP.NET and ASP.NET MVC framework," 14th International Conference on Computer and Information Technology (ICCIT 2011), Dhaka, Bangladesh, 2011, pp. 497-502, doi: 10.1109/ICCITechn.2011.6164840.

17. K. Zhang, Y. Guo, X. Wang, J. Yuan and Q. Ding, "Multiple Feature Reweight DenseNet for Image Classification," in IEEE Access, vol. 7, pp. 9872-9880, 2019, doi: 10.1109/ACCESS.2018.2890127.

18. Z. Xu, H. Yu, K. Zheng, L. Gao and M. Song, "A Novel Classification Framework for Hyperspectral Image Classification Based on Multiscale Spectral-Spatial Convolutional Network," 2021 11th Workshop on Hyperspectral Imaging and Signal Processing: Evolution in Remote Sensing (WHISPERS), Amsterdam, Netherlands, 2021, pp. 1-5, doi: 10.1109/WHISPERS52202.2021.9483998.

19. J. Feng, G. Bai, Z. Gao, X. Zhang and X. Tang, "Automatic Design Recurrent Neural Network for Hyperspectral Image Classification," 2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS, Brussels, Belgium, 2021, pp. 2234-2237, doi: 10.1109/IGARSS47720.2021.9554089.

20. Y. Liu, S. Zhang, F. Wang and N. Ling, "Tread Pattern Image Classification using Convolutional Neural Network Based on Transfer Learning," 2018 IEEE International Workshop on Signal Processing Systems (SiPS), Cape Town, South Africa, 2018, pp. 300-305, doi: 10.1109/SiPS.2018.8598400.

21. Z. Huang, X. Zhu, M. Ding and X. Zhang, "Medical Image Classification Using a Light-Weighted Hybrid Neural Network Based on PCANet and DenseNet," in IEEE Access, vol. 8, pp. 24697-24712, 2020, doi: 10.1109/ACCESS.2020.2971225.

22. H. Rouabeh, C. Abdelmoula and M. Masmoudi, "Performance evaluation of Decision Tree and neural network techniques for road scene image classification task," International Image Processing, Applications and Systems Conference, Sfax, Tunisia, 2014, pp. 1-6, doi: 10.1109/IPAS.2014.7043274.

23. V. Tiwari, C. Pandey, A. Dwivedi and V. Yadav, "Image Classification Using Deep Neural Network," 2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), Greater Noida, India, 2020, pp. 730-733, doi: 10.1109/ICACCCN51052.2020.9362804.

24. T. A. Youssef, B. Aissam, D. Khalid, B. Imane and J. E. Miloud, "Classification of chest pneumonia from x-ray images using new architecture based on ResNet," 2020 IEEE 2nd International Conference on Electronics, Control, Optimization and Computer Science (ICECOCS), Kenitra, Morocco, 2020, pp. 1-5, doi: 10.1109/ICECOCS50124.2020.9314567.

25. J. Zhang et al., "Detection and Classification of Pneumonia from Lung Ultrasound Images," 2020 5th International Conference on Communication, Image and Signal Processing (CCISP), Chengdu, China, 2020, pp. 294-298, doi: 10.1109/CCISP51026.2020.9273469.