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, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2023 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

ДОСЛІДЖЕННЯ МОДЕЛЕЙ НЕЙРОННОЇ МЕРЕЖІ RESNET ДЛЯ КЛАСИФІКАЦІЇ ОБ'ЄКТІВ НА ФОТОГРАФІЯХ

Спеціальність 122 «Комп'ютерні науки»

122 – БКР – 402.21910204

Виконав студент 4-го курсу, групи 402

_____
(підпис, ініціали та прізвище)

«___» _____ 202_ р.

Керівник: канд. техн. наук, доцент Сіденко Є.В.
(наук. ступінь, вчене звання)

_____
(підпис, ініціали та прізвище)

«___» _____ 202_ р.

Миколаїв – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**

Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)

Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем, докт. техн. наук, проф.

_____ Ю. П. Кондратенко

«____» _____2022р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Видано студенту групи 402 факультету комп'ютерних наук Герасу Олександру Миколайовичу.

1. Тема кваліфікаційної роботи: «Дослідження моделей нейронної мережі ResNet для класифікації об'єктів на фотографіях».

Керівник роботи Сіденко Євген Вікторович, к.т.н., доцент.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «___» ____ 20__ р. № _____

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

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

Очікуваний результат роботи: вебзастосунок для класифікації об'єктів 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.

6. Завдання до спеціальної частини: «Дії підчас артилерійського удару».

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	Боженко А.Л. викладач	

Керівник роботи _____к.т.н., доцент Сіденко Євген Вікторович_____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання _____Герас О.М._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «_23_» _____листопада____ 2022 р.

КАЛЕНДАРНИЙ ПЛАН

виконання бакалаврської кваліфікаційної роботи

Тема: Дослідження моделей нейронної мережі ResNet для класифікації об’єктів на фотографіях.

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників БКР	27.10.2022	30.10.2022	
2	Отримання завдання на виконання БКР	8.11.2022	10.11.2022	
3	Складання календарного плану роботи на весь період виконання БКР	1.12.2022	1.12.2022	
4	Отримання завдання на переддипломну практику	12.03.2023	12.03.2023	
5	Проходження переддипломної практики, збір та аналіз матеріалів до БКР	01.05.2023	14.05.2022	
6	Розробка звіту з переддипломної практики	15.05.2023	17.05.2023	
7	Піготовка даних, для розробки вебзастосунку та навчання нейромереж	18.05.2023	21.05.2023	
8	Створення нейромереж	22.05.2023	9.06.2023	
9	Створення вебзастосунку та тестування всієї системи	11.06.2023	19.06.2023	
10	Попередній захист БКР на засіданні комісії кафедри	29.05.2023	30.05.2023	
11	Доробка та остаточне оформлення БКР	02.06.2023	19.06.2023	
12	Подання БКР рецензенту	15.06.2023	17.06.2023	
13	Подання БКР, її електронної копії та інших документів (відгуку, рецензії) до захисту	19.06.2023	22.06.2023	
14	Захист БКР перед екзаменаційною комісією (ЕК)	26.06.2023	29.06.2023	

Розробив студент _____Герас О.М._____ _____
(прізвище та ініціали) *(підпис)*

Керівник роботи __к.т.н., доцент Сіденко Євген Вікторович__ _____
(наук. ступінь, вчене звання, прізвище та ініціали) *(підпис)*

«____» _____ 2023 р.

АНОТАЦІЯ

бакалаврської кваліфікаційної роботи студента групи 402 ЧНУ ім. Петра Могили

Гераса Олександра Миколайовича

Тема: «Дослідження моделей нейронної мережі ResNet для класифікації об'єктів на фотографіях»

Дослідження моделей нейронної мережі ResNet для класифікації об'єктів на фотографіях є актуальним у галузі комп'ютерного зору та обробки зображень. Використання нейронних мереж для класифікації стало широко поширеним, і ResNet є однією зі значних архітектур глибоких нейронних мереж, яка розв'язує проблему зникнення градієнту та дозволяє тренувати глибокі моделі з багатьма шарами.

Дослідження моделей ResNet має практичне значення, оскільки може покращити точність та ефективність класифікації зображень. Особливості ResNet, такі як використання зворотних з'єднань, роблять його привабливим для дослідження. Результати дослідження допоможуть впроваджувати ResNet в різних сферах, включаючи медицину, автоматичне розпізнавання образів та інші області, де класифікація об'єктів на фотографіях є важливим завданням.

Об'єкт роботи – класифікації літаків за допомогою нейронних мереж.

Предмет роботи – нейронні мережі для класифікації літаків на фотографіях.

Метою бакалаврської кваліфікаційної роботи є підвищення точності та класифікації об'єктів на фотографіях за рахунок застосування моделей нейронної мережі ResNet. Основними цілями дослідження є вивчення принципів функціонування ResNet, аналіз його особливостей порівняно з іншими архітектурами глибоких нейронних мереж, оцінка його ефективності та точності у класифікації об'єктів на фотографіях.

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

Отримані результати дослідження сприятимуть зрозумінню можливостей та обмежень ResNet 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 БКР.

У другому розділі досліджено основні принципи роботи нейронних мереж, що використовуватимуться в БКР, 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 літаків на фотографіях з використанням нейромережевих техноло гій з точністю близько 94%.

Бакалаврська кваліфікаційна робота містить 73 сторінок, 52 рисунок, 24 використаних джерел та 1 додаток.

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

ABSTRACT

for bachelor’s qualification work of a student of 402 group at Petro Mohyla Black Sea National University

Heras Oleksandr Mykolayovych

Topic: "Investigation of the models of the ResNet neural network for the classification of objects in photographs"

The study of the models of the ResNet neural network for the classification of objects in photographs is relevant in the field of computer vision and image processing. The use of neural networks for classification has become widespread, and ResNet is one of the significant deep neural network architectures that solves the gradient vanishing problem and allows training deep models with many layers.

Studying the performance of ResNet is of practical importance because it can improve the accuracy and efficiency of image classification. Features of ResNet, such as the use of backward connections, make it attractive for research. The results of the study will help to implement ResNet in various fields, including medicine, automatic pattern recognition and other areas where the classification of objects in photographs is an important task.

The object of the work is the classification of aircraft using neural networks.

The subject of the work is neural networks for the classification of aircraft in photographs.

The aim of the bachelor's qualification work is to increase the accuracy and classification of objects in photographs due to the use of ResNet neural network models. The main goals of the research are the study of the principles of ResNet functioning, the analysis of its features compared to other architectures of deep neural networks, the evaluation of its efficiency and accuracy in the classification of objects in photographs.

The obtained results of the study will contribute to the understanding of ResNet's capabilities and limitations 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.

In the first section, technologies for classifying objects in the photo, analysis of similar systems and formation of tasks for BW are considered.

The second chapter explores the basic principles of neural networks that will be used in BW, their architecture and learning algorithms.

The third chapter describes the process of data formation for training neural networks, their processing and sorting. The process of creating a web application for working with photo classification was also designed.

The fourth chapter presents the results of creating neural networks, creating a web application and testing them.

As a result, a system for classifying airplanes in photographs was developed using neural network technologies with an accuracy of about 94%.

Bachelor qualification thesis contains 73 pages, 52 figures, 24 used sources and 1 addition.

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

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

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;

ШІ – штучний інтелект;

SGD – Stochastic Gradient Descent;

DNN – deep neural network;

VGG – Visual Geometry Group;

JSON – JavaScript Object Notation.

ВСТУП

У сучасному світі, де величезна кількість інформації надходить кожної секунди, автоматична обробка даних із застосуванням штучного інтелекту стала надзвичайно актуальною. У цьому контексті нейронні мережі є потужним інструментом для розв'язання багатьох завдань з обробки даних, включаючи класифікацію об'єктів на фотографіях.

Одним з найбільш успішних підходів до класифікації зображень є використання нейронних мереж глибокого навчання, таких як ResNet. Вони здатні ефективно виявляти складні залежності в даних та забезпечувати високу точність класифікації.

Метою бакалаврської кваліфікаційної роботи є підвищення точності та класифікації об'єктів на фотографіях за рахунок застосування моделей нейронної мережі ResNet. Робота містить аналіз засобів та методів, що використовуються для побудови та тренування нейронних мереж глибокого навчання, огляд архітектури ResNet, а також практичну реалізацію дослідження на основі даних з відкритих джерел.

Результати цієї роботи можуть бути корисними для вивчення методів роботи з нейронними мережами глибокого навчання, а також для покращення точності класифікації об'єктів на фотографіях у різних областях застосування, включаючи медицину, транспорт, рекламу та бізнес-аналітику.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ. ПОСТАНОВКА ЗАДАЧІ

1.1 Основні поняття та визначення

Одним з головних понять, що є головною складовою бакалаврської кваліфікаційної роботи є computer vision, воно відноситься до галузі вивчення та дослідження, в якій комп'ютери навчаються інтерпретувати та розуміти візуальну інформацію з навколишнього світу. Це передбачає використання алгоритмів і методів машинного навчання для аналізу та обробки цифрових зображень і відео з метою отримання значущої інформації [2].

Computer vision має широкий спектр практичних застосувань, включаючи розпізнавання зображень і відео, виявлення та відстеження об'єктів, розпізнавання облич, автономні транспортні засоби, аналіз медичних зображень і багато іншого. Зі збільшенням доступності цифрових зображень і відео, а також швидким розвитком потужного комп'ютерного обладнання та програмного забезпечення комп'ютерний зір стає все більш важливою сферою з численними застосуваннями в промисловості, дослідженнях і повсякденному житті [3].

Computer vision потребує великої кількості даних. Він проводить аналіз даних знову і знову, доки не розпізнає відмінності та остаточно розпізнає зображення. Наприклад, щоб навчити комп'ютер розпізнавати автомобільні шини, йому потрібно надати величезну кількість зображень шин і предметів, пов'язаних із шинами, щоб дізнатися про відмінності та розпізнати шину, особливо без дефектів [3, 4].

Ось кілька прикладів усталених завдань комп'ютерного зору:

– класифікація зображень бачить зображення та може його класифікувати (собака, яблуко, обличчя людини). Точніше, він здатний точно передбачити, що дане зображення належить до певного класу. Наприклад, компанія соціальних медіа може використовувати його для автоматичної

ідентифікації та відокремлення небажаних зображень, завантажених користувачами;

– виявлення об’єктів може використовувати класифікацію зображень для ідентифікації певного класу зображень, а потім виявлення їх появи на зображенні чи відео. Приклади включають виявлення пошкоджень на складальній лінії або ідентифікацію обладнання, яке потребує технічного обслуговування;

– відстеження об’єкта стежить за об’єктом після його виявлення. Це завдання часто виконується за допомогою послідовних зображень або відео в реальному часі. Автономні транспортні засоби, наприклад, повинні не тільки класифікувати та виявляти такі об’єкти, як пішоходи, інші автомобілі та дорожня інфраструктура, вони повинні відстежувати їх у русі, щоб уникнути зіткнень і дотримуватися правил дорожнього руху;

– пошук зображень на основі вмісту використовує комп’ютерний зір для перегляду, пошуку та отримання зображень із великих сховищ даних на основі вмісту зображень, а не пов’язаних із ними тегів метаданих. Це завдання може включати автоматичне анотування зображення, яке замінює ручне тегування зображення. Ці завдання можна використовувати для систем управління цифровими активами та можуть підвищити точність пошуку та вилучення.

Кількість фото, необхідних для створення нейромережі для класифікації зображень, залежить від кількості різних параметрів, таких як складність класифікації, кількість категорій, кількість елементів у кожній категорії тощо [5,6,7].

У загальному випадку, рекомендується мати принаймні 100 зображень кожної категорії для досягнення гарної точності класифікації. Таким чином, для створення нейромережі для класифікації 4 типів зображень мінімальна кількість фото становитиме 400 (100 зображень на кожен тип). В нашому випадку використано по 250 фотографій для літаків кожного типу.

Головним будівельним матеріалом всіх нейромереж є нейрон - це базовий елемент штучної нейронної мережі. Він приймає вхідні дані, обчислює деякі дії з ними і генерує вихідні дані. Нейрон складається з декількох компонентів, включаючи вхідні зв'язки, зважування, функцію активації та вихідний сигнал.

Кожен нейрон отримує вхідні дані через свої вхідні зв'язки, де кожен вхідний зв'язок має свою вагу. Потім нейрон обчислює зважену суму всіх вхідних зв'язків та їх відповідних ваг, і ця сума передається через функцію активації, що здійснює нелінійне перетворення вхідних даних. Результат цієї функції стає вихідним значенням нейрона, яке може передаватися іншим нейронам у мережі для подальшого оброблення.

Нейрони в штучних нейронних мережах використовуються для вирішення різноманітних завдань, включаючи класифікацію, розпізнавання образів, машинний переклад та багато іншого [7].

Головним об’єктом вивчення є ResNet, скорочення від «Residual Network» — це архітектура глибокої нейронної мережі, яка була представлена в 2015 році дослідниками Microsoft Research. Це тип згорткової нейронної мережі (CNN), яка розроблена так, щоб бути набагато глибшою, ніж традиційні CNN, і водночас вирішує проблему зникаючих градієнтів, які можуть виникнути під час навчання дуже глибоких мереж [6].

Моделі глибокого навчання навчаються на великих наборах помічених даних з використанням нейронних мереж, що містять безліч прошарків навчання.

Глибоке навчання:

– краще вирішує деякі завдання;

– потребує величезних обсягів даних на навчання.

Класифікація зображень — це конкретне завдання їхньої автоматичної класифікації за категоріями, наприклад:

– визначення того, чи містить зображення людське обличчя;

– для розрізнення зображень котів та собак.

Ключовим нововведенням ResNet є використання залишкових з'єднань, які дозволяють інформації з попередніх рівнів обходити наступні рівні та надходити безпосередньо до виходу. Це допомагає гарантувати, що градієнти можуть вільно проходити через мережу під час навчання, полегшуючи навчання дуже глибоких моделей.

Доведено, що ResNet дуже ефективний для широкого спектру завдань комп'ютерного зору, включаючи класифікацію зображень, виявлення об'єктів і сегментацію. Фактично, він був використаний як основа для багатьох найсучасніших моделей комп'ютерного зору та допоміг підняти продуктивність цих моделей на нові висоти [5].

Було запропоновано кілька типів архітектур ResNet, кожна з яких має різну кількість рівнів і структур. Деякі з найбільш часто використовуваних моделей ResNet описані нижче.

1. ResNet-18: це відносно невелика архітектура ResNet з 18 шарами, включаючи залишкові блоки з 2-3 шарами згортки кожен.

2. ResNet-34: це трохи більша архітектура ResNet із 34 шарами, включаючи залишкові блоки з 3-4 шарами згортки кожен.

3. ResNet-50: це популярна архітектура ResNet із 50 шарами, включаючи залишкові блоки з трьома згортковими шарами на кожному. Це була перша архітектура, яка досягла найсучаснішої продуктивності в задачі класифікації ImageNet.

4. ResNet-101: це більша архітектура ResNet зі 101 шаром, включаючи залишкові блоки з трьома згортковими шарами кожен.

5. ResNet-152: це найбільша архітектура ResNet із 152 шарами, включаючи залишкові блоки з трьома згортковими шарами кожен. Зазвичай він

використовується для дуже складних завдань комп'ютерного зору, які потребують високого рівня точності.

ML.NET — це міжплатформна бібліотека машинного навчання з відкритим вихідним кодом, розроблена Microsoft, яка дозволяє розробникам створювати користувацькі моделі машинного навчання за допомогою мов програмування .NET, таких як C# і F# [14, 15].

ML.NET надає низку інструментів і бібліотек, які спрощують створення, навчання та розгортання моделей машинного навчання для різноманітних програм, включаючи аналіз настроїв, виявлення шахрайства, класифікацію зображень тощо. Він включає в себе широкий спектр попередньо створених алгоритмів і моделей, які можна використовувати з набору сценаріїв, а також можливість налаштовувати та розширювати ці моделі відповідно до конкретних потреб [15].

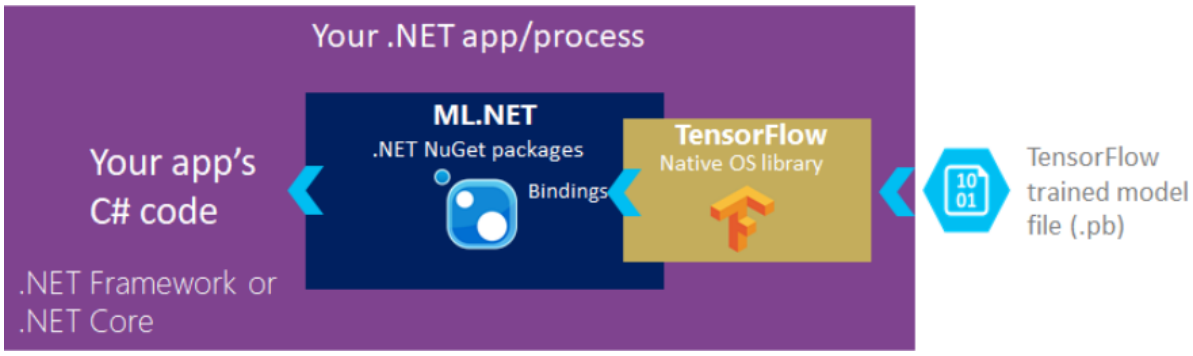


Рисунок 1.1 - Схема роботи ML.Net

Однією з ключових особливостей ML.NET є його інтеграція з екосистемою .NET, що дозволяє розробникам використовувати знайомі мови програмування та інструменти, а також легко включати можливості машинного навчання у свої існуючі програми. Він також включає підтримку популярних джерел даних, таких як SQL Server, Azure Storage та Apache Spark, що полегшує роботу з великими наборами даних.

Keras — це бібліотека нейронних мереж високого рівня з відкритим кодом, написана на Python. Він був розроблений Франсуа Шолле та був спочатку випущений у 2015 році. Keras пропонує простий і ефективний спосіб побудови та навчання нейронних мереж, що робить його популярним вибором для дослідників і практиків глибокого навчання [17,18].

Keras створювалася як гнучка модульна бібліотека, яку легко налаштовувати та модифікувати. Вона безкоштовна, має відкритий вихідний код, який може подивитися будь-хто.

Назва читається як «Керас», що в перекладі з грецької означає «ріг». Це посилання до рядків з «Одіссеї».

Keras має вузьку спеціалізацію. Це інструмент для фахівців з машинного навчання, які працюють з мовою Python: саме його найчастіше використовують завдяки зручності математичних обчислень. Keras застосовують розробники, які створюють, налаштовують та тестують системи машинного навчання та штучного інтелекту, насамперед нейронні мережі.

Однією з ключових переваг Keras є простота використання та зручний API, який дозволяє розробникам швидко створювати та експериментувати з моделями нейронних мереж. Він забезпечує простий інтерфейс для побудови та навчання різних типів архітектур нейронних мереж, таких як згорткові нейронні мережі (CNN), рекурентні нейронні мережі (RNN) тощо.

Keras відносно легко освоїти та працювати з ним, тому що він забезпечує інтерфейс Python з високим рівнем абстракції, маючи можливість використовувати кілька серверних інтерфейсів для обчислень. Це робить Keras повільнішим, ніж інші фреймворки глибокого навчання, але надзвичайно зручним для початківців.

Keras також підтримує кілька бекендів, зокрема TensorFlow, CNTK і Theano, що дозволяє користувачам вибрати найкращий бекенд для своїх конкретних потреб.

Деякі з ключових функцій Keras включають:

– Keras надає простий та інтуїтивно зрозумілий API для створення моделей, що дозволяє легко експериментувати з різними архітектурами;

– інструменти попередньої обробки: Keras містить низку інструментів для попередньої обробки даних, таких як збільшення даних зображення, заповнення послідовності та одноразове кодування;

– Keras підтримує різноманітні архітектури нейронних мереж, включаючи CNN, RNN, автокодери тощо;

– інтеграція з TensorFlow, Keras є частиною проекту TensorFlow, який надає потужний і гнучкий сервер для навчання нейронних мереж.

Мета Keras — надати перевагу будь-якому розробнику, який хоче постачати програми на основі машинного навчання. Keras зосереджується на швидкості налагодження, елегантності та лаконічності коду, зручності обслуговування та розгортанні. Коли обираємо Keras, код стає меншим, легшим для читання, легшим для ітерації. Моделі працюють швидше завдяки компіляції XLA та оптимізації Autograph, і їх легше розгортати на будь-якій поверхні (сервер, мобільний пристрій, браузер, вбудований) завдяки TF Serving, TF Lite та TF.js.

Що робити якщо точність начання занизька? Якщо точність моделі недостатньо висока, можна зробити таке:

– збільшити час навчання. Чим довше триває навчання, тим більше алгоритмів та параметрів може випробувати система машинного навчання;

– додати більше даних. Іноді обсяг даних недостатній для навчання високоякісної моделі машинного навчання. Це особливо справедливо для наборів даних із невеликою кількістю прикладів;

– збалансувати дані. Для завдань класифікації навчальний набір має бути збалансований за всіма категоріями. Наприклад, якщо є чотири класи для 100 зразків, причому перші два класи (тег1 і тег2) використовуються для 90 записів,

інші два (тег3 і тег4) — для решти 10 записів, незбалансованість даних може позначитися на правильності прогнозування класів тег3 і тег4.

1.2 Огляд та аналіз наявних аналогів та публікацій

Одним з аналогів системи для класифікації літаків є сайт Jetphotos. На сайті можна завантажити сфотографований літак та завантажити цього. Серед недоліків, це те що фото повинне буде заданого розширення та не більше ніж 250 Кб, тож потрібно буде виконати певні маніпуляції з фото, щоб воно відповідало вимогам сайту.

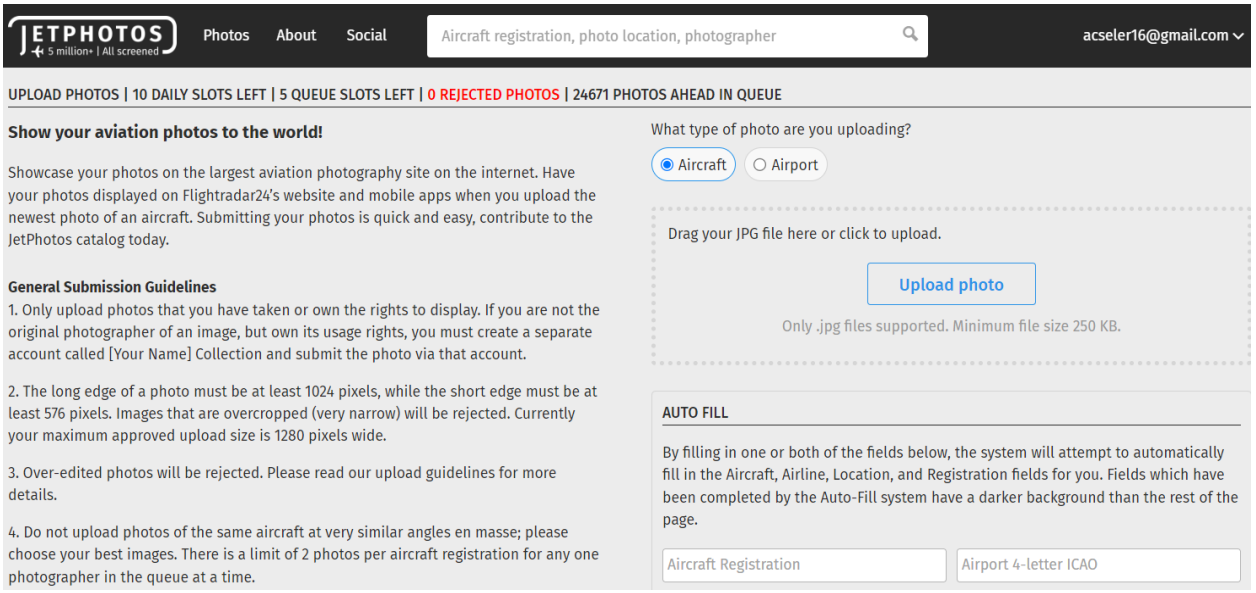


Рисунок 1.2 - Сторінка на сайті JetPhotos для завантаження фото літака

Серед переваг є можливість розпізнати абсолютно будь який літак, але сам процес впізнання займає багато часу, а також фото проходить модерацію перед тим як воно буде відображене користувачам сайту.

Наступним варіантом є сайт для розпізнавання об’єктів від компанії Aspose. На відміну від системи створеної в ході виконання бакалаврської кваліфікаційної роботи, а саме системи класифікації об’єкту на зображенні, даний сайт займається

виключно розпізнаванням об’єктів на фото, тож отримати детальну назву літака неможливо, вдасться лише перевірити чи зображений на фото літак чи це, щось інше.

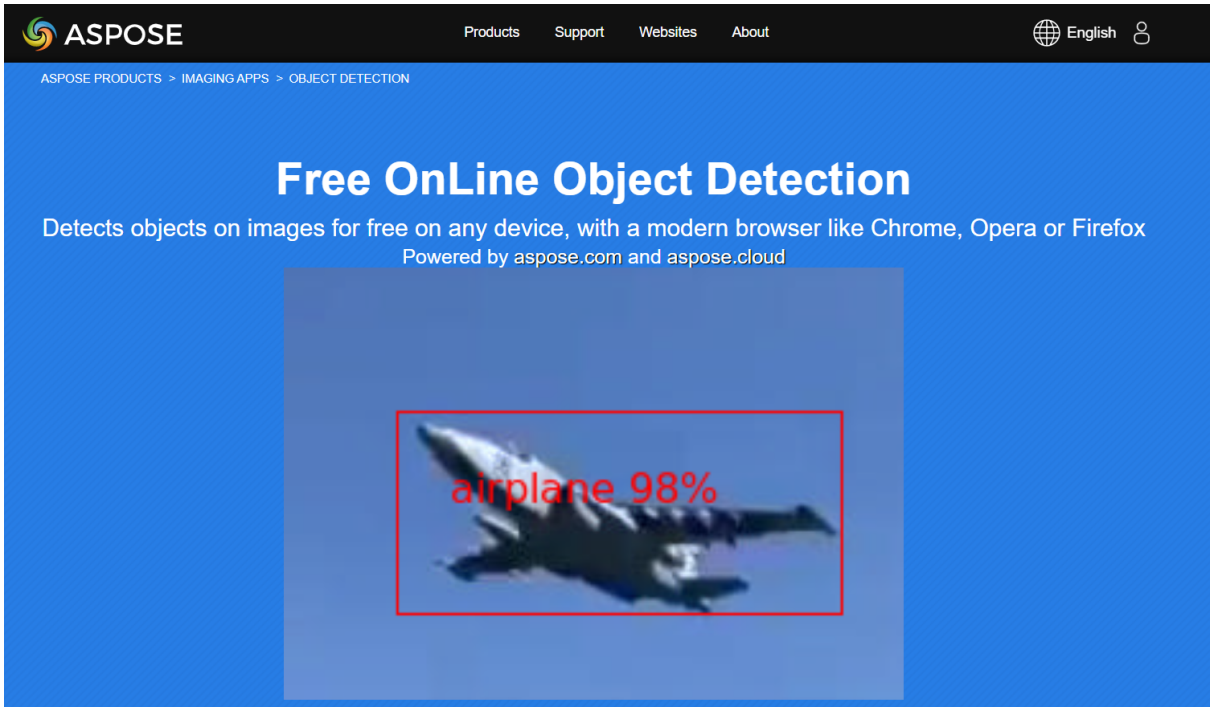


Рисунок 1.3 - Сторінка за класифікованим літаком на сайті Aspose

Якщо говорити про типи архітектур нейронних мереж, то нижче виділимо основні архітектури.

Прямі нейронні мережі - це тип нейронних мереж, в яких інформація поширюється тільки в одному напрямку - від вхідних вузлів до вихідних. Така мережа складається зі шарів вузлів, де кожен вузол зв'язаний з кожним вузлом на попередньому та наступному шарах. Кожен вузол зв'язаний з вузлами попереднього шару через зважені зв'язки, які передають сигнали від одного вузла до наступного. Кожен вузол обробляє сигнали, які йому надійшли від попереднього шару, застосовуючи певну активаційну функцію. Вихідний шар мережі містить вузли, які повертають результат обчислення мережі [3].

Рекурентні нейронні мережі - це тип нейронних мереж, які мають здатність використовувати інформацію з попередніх вхідних даних для обробки поточного вхідного сигналу. На відміну від прямих нейронних мереж, де інформація поширюється тільки в одному напрямку, в RNN інформація може поширюватись в обидва напрямки: від вхідних вузлів до вихідних та від вихідних вузлів до попередніх шарів.

Рекурентні нейронні мережі мають спеціальний тип вузлів, які називаються рекурентними вузлами. Кожен рекурентний вузол має внутрішню пам'ять, що дозволяє мережі зберігати інформацію з попередніх вхідних даних та використовувати цю інформацію для подальшої обробки поточного вхідного сигналу. Рекурентні вузли зв'язані між собою, що дозволяє мережі використовувати внутрішню пам'ять для обробки послідовних вхідних даних.

Автокодувальні нейронні мережі - це тип нейронних мереж, які використовуються для зменшення розміру та зменшення шуму вхідних даних, зберігаючи при цьому інформацію, яка є необхідною для їх відновлення.

Автокодувальна мережа має дві основні частини: кодер та декодер. Кодер приймає вхідні дані і перетворює їх в менший розмір за допомогою складного шару нейронів. Потім цей код передається декодеру, який використовує його для відновлення вихідних даних.

В процесі навчання автокодувальної мережі використовуються вхідні дані, і мережа намагається знайти оптимальний спосіб стиснення цих даних у кодері та їх відновлення в декодері. Часто в процесі навчання застосовуються методи зменшення розміру вихідних даних з метою зміни їх форми та зменшення шуму в даних.

Нейронні мережі з довільною архітектурою - це нейронні мережі, які можуть мати будь-яку архітектуру залежно від вимог задачі.

У звичайних нейронних мережах, таких як персептрон або згорткові нейронні мережі, архітектура визначається заздалегідь і вона має фіксований набір шарів та кількість нейронів у кожному з них. Але в нейронних мережах з довільною архітектурою ці обмеження відсутні. Вони можуть мати будь-яку кількість шарів та нейронів, залежно від вимог задачі.

Такі мережі використовуються в задачах, де стандартні архітектури нейронних мереж не вистачає для досягнення бажаного рівня точності. Наприклад, вони можуть бути використані для розпізнавання складних об'єктів у зображеннях або для вирішення складних завдань управління рухом роботів.

Глибокі нейронні мережі - це нейронні мережі, які складаються з багатьох шарів обчислювальних вузлів, кожен з яких виконує певну операцію з даними. Ці мережі використовуються для розв'язання складних завдань у різних областях, таких як обробка зображень, розпізнавання мови, природні мови та багато інших [21, 22].

Одна з особливостей глибоких нейронних мереж полягає в тому, що кожен наступний шар використовує вихідні дані попереднього шару для отримання більш складних функцій. Це дозволяє мережі розпізнавати більш складні патерни в даних.

Навчання глибоких нейронних мереж вимагає значної кількості даних та обчислювальних ресурсів. Для зменшення кількості параметрів та покращення ефективності навчання можуть використовуватися такі методи, як згорткові нейронні мережі та рекурентні нейронні мережі.

Нейронні мережі з підсиленням - це клас нейронних мереж, які використовуються для вирішення завдань з управлінням поведінкою агента у визначеному середовищі.

Ці мережі використовують алгоритми навчання з підсиленням, які базуються на концепції нагород та покарань. Агент знаходиться в деякому середовищі і

приймає рішення про свої дії з метою отримання максимальної нагороди. Нейронна мережа з підсиленням навчається шляхом програвання ігор у середовищі і на основі отриманих нагород покращує свої дії в подальшому.

Генеративні згорткові мережі - це клас нейронних мереж, які комбінують в собі згорткові шари зі здатністю генерації нових зображень.

GCN зазвичай використовуються для генерації зображень за допомогою навчання без вчителя. Їх головна ідея полягає в тому, що мережа навчається виділяти важливі особливості зображення та відтворювати їх у новому зображенні [8].

GCN зазвичай складаються з енкодера та декодера. Енкодер згортає вхідне зображення у меншу кількість прихованих ознак, тоді як декодер відтворює ці ознаки у вихідному зображенні. Цей процес навчання може бути використаний для створення нових зображень, які не були використані під час навчання.

Мережі з відповідями з короткими повідомленнями - це клас нейронних мереж, які використовуються для вирішення завдань з обробки природньої мови, зокрема для відповіді на запитання та взаємодії з користувачами.

SMRN зазвичай складаються з двох компонентів: енкодера та декодера. Енкодер приймає на вхід запитання користувача та перетворює його у вектор прихованого стану. Декодер використовує цей вектор для генерації короткої відповіді на запитання.

Оскільки SMRN здатні працювати зі словами та фразами, вони можуть використовуватись для вирішення завдань розмовного та текстового пошуку, рекомендацій, відповіді на запитання, чат-ботів та інших завдань з обробки природньої мови.

Мережі з підтримкою пам'яті - це клас нейронних мереж, які мають можливість зберігати та доступатися до інформації в пам'яті під час обробки

вхідних даних. Ці мережі використовують пам'ять, щоб зберігати контекстну інформацію та взаємодіяти з нею під час обробки нової інформації [5].

Одним з найпопулярніших прикладів мереж з підтримкою пам'яті є NTM, який був запропонований у 2014 році. NTM включає механізм для зберігання та доступу до великої кількості даних, які можуть бути використані для розв'язання завдань з обробки природньої мови, машинного перекладу, розпізнавання образів та інших завдань, які вимагають доступу до контекстної інформації.

1.3 Постановка задачі

Об'єкт роботи – класифікації літаків за допомогою нейронних мереж.

Предмет роботи – нейронні мережі для класифікації літаків на фотографіях.

Метою бакалаврської кваліфікаційної роботи є підвищення точності та класифікації об'єктів на фотографіях за рахунок застосування моделей нейронної мережі ResNet. Нижче описані головні завдання дослідження:

— проаналізувати існуючі методи та алгоритми класифікації зображень, включаючи нейронні мережі;

— вивчити теоретичні основи нейронної мережі ResNet, її архітектуру та особливості роботи;

— зібрати та підготувати набір даних для класифікації зображень;

— навчити нейронну мережу ResNet на підготовленому наборі даних та оцінити її ефективність;

— здійснити експерименти для визначення можливості покращення результатів класифікації, зокрема за допомогою технік, таких як transfer learning та fine-tuning;

— зробити висновки про ефективність та можливості покращення роботи нейронної мережі ResNet для класифікації об'єктів на фото та визначити перспективи подальшого дослідження.

Висновки до розділу 1

Тема дослідження роботи нейронної мережі ResNet для класифікації об'єктів на фото є дуже актуальною в сучасному світі, де збільшується кількість зображень, що потребують автоматичної обробки та класифікації. Нейронні мережі ResNet є одними з найбільш ефективних та широко використовуваних мереж для класифікації зображень, тому дослідження їх моделей є дуже важливим для розвитку області комп'ютерного зору та машинного навчання.

В першому розділі бакалаврської кваліфікаційної роботи було розглянуто основні поняття та визначення, пов'язані з дослідженням моделей нейронної мережі ResNet для класифікації об'єктів на фото. Було проведено аналіз всіх аналогічних систем, що використовуються для класифікації об'єктів на фото, та було виявлено переваги та недоліки кожної з них.

Однією з основних задач бакалаврської кваліфікаційної роботи є дослідження моделей нейронної мережі ResNet, яка є однією з найбільш ефективних та широко використовуваних мереж для класифікації зображень. Задача полягатиме у дослідженні моделей мережі на різних наборах даних та знаходженні оптимальних параметрів мережі для досягнення максимальної точності класифікації.

Завдяки проведеному аналізу та сформованій задачі, можна очікувати, що дослідження роботи нейронної мережі ResNet буде важливим та актуальним для розвитку області комп'ютерного зору та машинного навчання. Результати дослідження можуть бути корисні для покращення систем класифікації зображень та для подальшого застосування у різних областях, таких як медицина, промисловість та безпека.

2 НЕЙРОМЕРЕЖЕВІ ТЕХНОЛОГІЇ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ ДОСЛІДЖЕННЯ МОДЕЛЕЙ НЕЙРОННОЇ МЕРЕЖІ RESNET ДЛЯ КЛАСИФІКАЦІЇ ОБ'ЄКТІВ НА ЗОБРАЖЕННЯХ

2.1 Модель машинного навчання ML.Net

ML.NET — це платформа машинного навчання для розробників .NET; можна використовувати ML.NET для інтеграції моделей машинного навчання у програми .NET. Ви можете використовувати ML.NET для багатьох сценаріїв, таких як аналіз настроїв, прогнозування цін, рекомендації продукту, прогнозування продажів, класифікація зображень, виявлення об'єктів тощо.

В чому різниця між машинним навчанням та штучним інтелектом? Штучний інтелект — це галузь обчислювальної техніки, яка передбачає навчання комп'ютерів виконувати дії, які зазвичай потребують людського інтелекту. Машинне навчання — це підмножина штучного інтелекту, яка включає в себе комп'ютери, які вивчають і знаходять закономірності в даних, щоб потім самостійно робити прогнози щодо нових даних [2].

Корпорація Майкрософт пропонує багато продуктів і послуг AI і ML, тому нижче описано розподіл відмінностей між ними.

1. ML.NET: створює спеціальні рішення машинного навчання та інтегрує їх у програми .NET.

2. Azure Cognitive Services: хмарні служби, які надають попередньо створені моделі ШІ та машинного навчання для додавання до програм. Включає набір API для використання різноманітних моделей природних методів спілкування за допомогою зору та мови.

3. Azure Machine Learning: комплексне середовище в хмарі для розміщення наскрізного життєвого циклу моделі машинного навчання, включаючи навчання моделі, керування версіями, розгортання та керування випусками в хмарному масштабі.

ML.NET дозволяє створювати моделі машинного навчання для різних завдань, включаючи класифікацію, регресію, кластеризацію, аналіз аномалій та інше [13].

Нижче наведені приклади сценаріїв роботи ML.NET:

– можна використовувати ML.NET для класифікації текстів на основі їх змісту, наприклад, для автоматичної класифікації новин або відгуків на продукти;

– ML.NET дозволяє використовувати алгоритми машинного навчання для аналізу великих обсягів даних, наприклад, для виявлення закономірностей та трендів в історичних даних;

– за допомогою ML.NET можна створити моделі машинного навчання для рекомендацій, наприклад, для рекомендацій товарів на основі попередніх покупок користувачів;

– можна використовувати ML.NET для кластеризації даних, наприклад, для групування користувачів за їх поведінкою на сайті;

– ML.NET дозволяє створювати моделі машинного навчання для розпізнавання образів, наприклад, для автоматичного розпізнавання номерних знаків на фото.

ML.NET — це безкоштовна міжплатформна бібліотека машинного навчання з відкритим кодом для платформи розробника .NET.

ML.NET дозволяє навчати, створювати та надсилати власні моделі машинного навчання за допомогою C# або F# для різноманітних сценаріїв ML. ML.NET містить такі функції, як автоматичне машинне навчання (AutoML), а також такі інструменти, як ML.NET CLI та ML.NET Model Builder, які спрощують інтеграцію машинного навчання у програми [16].

Можна використовувати ML.NET майже з будь-якою програмою .NET, включаючи вебпрограми та служби, мікросервіси/контейнери, настільні програми

(WPF і WinForms), функції Azure та будь-які серверні програми Azure, а також консольні програми.

ML.NET підтримується в .NET, .NET Core (версія 2.0 і вище) і .NET Framework (версія 4.6.1 і вище). Наразі ML.NET підтримується процесами x64 і x86. ML.NET є кросформним, тому він підтримується в macOS, Linux і Windows [12].

Стандартна для багатокласової класифікації — мікроточність. Чим ближче значення мікроточності до 100% або 1,0, тим краще [11].

ML.NET два типи метрики для багатокласової класифікації – макроточність та мікроточності. Подібно до мікроточності, чим ближче її значення до 1,0, тим краще. Нижче наведено найкращі визначення цих двох типів точності:

– мікроточність — як часто запит, що входить, передається відповідній команді співробітників?

– макроточність — як часто вхідний запит підходить для команди в середньому випадку?

DNN — це тип машинного навчання, який імітує спосіб навчання мозку. Його використовують для різноманітних завдань; деякі, як-от інструменти мовного перекладу та пошуку зображень, і деякі, як-от медична діагностика – UCLA навчив DNN виявляти ракові клітини.

Загальна ідея DNN полягає в тому, що вона навчається шляхом повторюваних дій із колекції зразків, як-от 100 зображень різних собак, на відміну від набору рукотворних правил, як-от «у собаки чорний ніс і вигнуті вуха». Таким чином, DNN навчається так само, як і людський мозок – через практику та помилки [21].

DNN в ml.net може бути навчена на даних, що містять числові та категоріальні ознаки. Після навчання можна використовувати модель для класифікації нових даних, які не були використані для навчання. Для навчання

DNN в ml.net можна використовувати різні алгоритми, такі як SGD, Adam, RMSprop тощо.

DNN в ml.net може бути створений та навчений за допомогою API-інтерфейсу, що дозволяє використовувати його в різних типах застосувань, таких як обробка зображень, розпізнавання мовлення, виявлення шахрайства тощо. Нижче описано принцип роботи DNN.

1. Комп’ютер отримує частину інформації, наприклад зображення чи звук. Скажімо, у нашому випадку ми маємо датасет з фотографіями літаків. На відміну від людей, комп’ютер не знає, що це таке.

2. Комп’ютер пропускає картинки літаків через свою DNN, розпізнаючи, що він може, і сортуючи його елементи, як-от чорний або білий кольори.

3. Коли він досягає кінця цього процесу, він вирішує, чи є обраний літак заданого типу чи ні.

4. Комп’ютер отримує зворотній зв’язок на цю відповідь – «так» чи «ні», який використовує для посилення прийняття рішень.

5. Процес повторюється знову і знову з великою кількістю різних фотографій літаків, поки комп’ютер не навчиться розпізнавати їх миттєво. Так само, як мозок.

Тепер припустімо, що цей DNN був навчений десятками тисяч фотографій літаків із реального життя – в аеропортах, полях чи в небі. DNN навчиться ідентифікувати та балансувати кожен літак усередині себе, щоб ви могли отримати доступ до найважливіших для вас типів літаків [10].

Отже, це саме те, що було зроблено. Було проведено навчання DNN з більш ніж однією тисячею фотографіями літаків, подібних до тих, які потім навчилися аналізувати та класифікувати.

2.2 Архітектура нейромереж ResNet-50

ResNet-50 — це варіант моделі ResNet, яка має 48 шарів Convolution, а також 1 Max Pool і 1 Average Pool. Він має 3,8✖10^9 операцій з плаваючою комою. Це широко використовувана модель ResNet [6,8].

ResNet-50 є глибокою нейронною мережею, яка містить 50 шарів. Основним елементом мережі є блок зі зв'язками "residual", який складається з декількох шарів. Нижче наведено повний перелік шарів, які входять до складу ResNet-50:

— вхідний шар - зображення;

— початковий конволюційний шар, який виконує першу обробку зображення;

— блок зі зв'язками "residual" - складається з двох конволюційних шарів та одного шару додавання;

— повторення 2-3 ще 3 рази, що дозволяє створити 4 блоки зі зв'язками "residual";

— після кожного блоку зі зв'язками "residual" застосовується макс-пулінг, що дозволяє зменшити розмір зображення та сконцентрувати увагу на найважливіших деталях;

— після останнього блоку зі зв'язками "residual" виконується повторно макс-пулінг;

— повний зв'язний шар (FC layer) - останній шар мережі, який відповідає за класифікацію зображення на певні категорії.

Блок зі зв'язками "residual" - це основний блок ResNet-50, який дозволяє мережі ефективно вирішувати проблему зниклих градієнтів (vanishing gradients) при тренуванні глибоких нейронних мереж [7].

Зазвичай, при тренуванні глибоких мереж, може виникати проблема зниклих градієнтів, коли градієнт стає настільки малим, що нейронна мережа перестає

навчатися. Це стається через те, що градієнт зникає під час проходження через багато шарів мережі.

Блок зі зв'язками "residual" дозволяє уникнути цієї проблеми шляхом використання зв'язку, який "перескакує" через декілька шарів. Замість того, щоб передавати вихідний сигнал з одного шару до наступного, блок зі зв'язками "residual" додає вихідний сигнал до виходу на певному етапі обробки внутрішнього блоку, що дозволяє "перенести" частину інформації на наступний шар.

ResNets спочатку застосовувалися для завдання розпізнавання зображень, але, як зазначено в документі, цю структуру також можна використовувати для завдань, не пов’язаних із комп’ютерним зором, щоб досягти кращої точності.

Можна заперечити, що просте складання кількох шарів також дає нам кращу точність, чому виникла потреба в залишковому навчанні для навчання надглибоких нейронних мереж [5, 6].

Цю архітектуру можна використовувати для завдань комп’ютерного зору, таких як класифікація зображень, локалізація об’єктів, виявлення об’єктів. Ця структура також може бути застосована до завдань, не пов’язаних із комп’ютерним баченням, щоб надати їм перевагу глибини та також зменшити витрати на обчислення.

Мінусами використання цієї моделі є проблема зникнення/вибуху градієнтів. Ці проблеми в основному вирішувались багатьма способами та дозволяли мережам із десятками рівнів зближуватися, але коли глибокі нейронні мережі починають збігатися, ми бачимо іншу проблему: точність насичується, а потім швидко погіршується та це не було спричинено переобладнанням, як можна здогадатися, а додавання додаткових шарів до відповідної глибокої моделі лише збільшило помилку навчання.

Цю проблему було додатково вирішено шляхом використання дрібнішої моделі та глибокої моделі, яка була створена з шарами дрібної моделі та додавання до неї шарів ідентичності, і, відповідно, глибша модель не повинна була спричинити більшу помилку навчання, ніж її аналог, оскільки додані шари були лише шарами ідентичності.

Для ResNet 50 і новіших версій була внесена невелика зміна: раніше підключення швидкого доступу пропускали два шари, але тепер вони пропускають три шари, а також було додано шари згортки 1 * 1, які ми збираємося детально розглянути з ResNet 50.

Отже, як ми бачимо (рисунок 2.1), архітектура ResNet-50 містить наступні елементи.

Згортка з розміром ядра 7 * 7 і 64 різних ядра, усі з кроком розміру 2, дає нам 1 шар. Далі ми бачимо максимальне об’єднання з розміром кроку 2.

У наступній згортці є ядро 1 * 1, 64, яке слідує за ядром 3 * 3, 64 і, нарешті, ядро 1 * 1, 256. Ці три шари повторюються загалом 3 рази, що дає нам 9 шарів на цьому кроці.

Далі ми бачимо ядро 1 * 1, 128, після цього ядро 3 * 3, 128 і ядро 1 * 1, 512. Цей крок було повторено 4 рази, що дало нам 12 шарів на цьому кроці.

Після цього є ядро 1 * 1, 256 і ще два ядра з 3 * 3, 256 і 1 * 1, 1024, і це повторюється 6 разів, даючи нам загалом 18 шарів.

А потім знову ядро 1 * 1, 512 з двома ядрами 3 * 3, 512 і 1 * 1, 2048, і це повторилося 3 рази, що дало нам загалом 9 шарів.

Після цього ми робимо середній пул і закінчуємо його повністю підключеним шаром, що містить 1000 вузлів, і в кінці функцію softmax, тож це дає нам 1 шар [10].

Ми фактично не враховуємо функції активації та максимальні/середні рівні об’єднання. Тож підсумовуючи це, ми отримуємо

$1 + 9 + 12 + 18 + 9 + 1 = 50$ шарів глибокої згорткової мережі. В результаті отримуємо оцінку максимально досяжних обчислювальних здібностей в 3.8×10^9 FLOPs [1].

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Рисунок 2.1 - Рисунок архітектур моделей ResNet

У контексті нейромереж "FLOPS" (floating point operations per second) відноситься до кількості операцій з плаваючою комою, які виконується в нейромережі за одну секунду. В термінах нейромережі, FLOPS можна використовувати як міру продуктивності обчислювального обладнання, такого як центральний процесор (CPU) або графічний процесор (GPU) [2].

2.3 Архітектура нейромереж ResNet-101

ResNet-101 - це глибока нейронна мережа, яка була запропонована у 2015 році в статті "Deep Residual Learning for Image Recognition" авторами Kaiming He, Xiangyu Zhang, Shaoqing Ren та Jian Sun з Microsoft Research Asia. ResNet-101 є однією з архітектур мереж ResNet, які засновані на підході з в икористанням

"residual connections", що дозволяє ефективно боротися з проблемою зниклих градієнтів в глибоких нейронних мережах.

ResNet101 має 101 шарів та є більш глибокою моделлю, ніж ResNet-50. Вона включає в себе дві основні блоки: "базовий блок" та "блок бутель". Базовий блок використовує дві згорткові шари та один шар Batch Normalization та додавання "residual connection". Блок бутель складається з трьох базових блоків, які виконують зменшення розміру зображення вдвічі та збільшення кількості каналів.

ResNet-101 є досить складною моделлю, яка вимагає великої кількості обчислювальних ресурсів для навчання та застосування. Вона зазвичай використовується для задач класифікації зображень на великих датасетах, таких як ImageNet [6].

Тепер розберемо архітектуру ResNet-101, вона в свою чергу містить наступі елементи (рисунок 2.1).

Згортка з розміром ядра 7 * 7 і 64 різних ядра, усі з кроком розміру 2, дає нам 1 шар. Далі ми бачимо максимальне об’єднання з розміром кроку 2.

У наступній згортці є ядро 1 * 1, 64, яке слідує за ядром 3 * 3, 64 і, нарешті, ядро 1 * 1, 256. Ці три шари повторюються загалом 3 рази, що дає нам 9 шарів на цьому кроці.

Далі ми бачимо ядро 1 * 1, 128, після цього ядро 3 * 3, 128 і ядро 1 * 1, 512. Цей крок було повторено 4 рази, що дало нам 12 шарів на цьому кроці.

Після цього є ядро 1 * 1, 256 і ще два ядра з 3 * 3, 256 і 1 * 1, 1024, і це повторюється 23 разів, даючи нам загалом 69 шарів.

А потім знову ядро 1 * 1, 512 з двома ядрами 3 * 3, 512 і 1 * 1, 2048, і це повторилося 3 рази, що дало нам загалом 9 шарів.

Після цього ми робимо середній пул і закінчуємо його повністю підключеним шаром, що містить 1000 вузлів, і в кінці функцію softmax, тож це дає нам 1 шар.

Ми фактично не враховуємо функції активації та максимальні/середні рівні об’єднання. Тож підсумовуючи це, ми отримуємо $1 + 9 + 12 + 69 + 9 + 1 = 101$ шарів глибокої згорткової мережі. В результаті отримуємо оцінку максимально досяжних обчислювальних здібностей в 7.6×10^9 FLOPs.

Після того як було визначено обчислювальні здібності нейромереж типу ResNet, нижче описано її покрокову роботу [4].

1. Початкове зображення подається на вхід нейронної мережі.
2. Перший шар мережі - це звичайний конволюційний шар, який виконує попередню обробку зображення.
3. Після цього виконується послідовне застосування блоків зі зв'язками "residual". Кожен такий блок містить два шари конволюції та один шар додавання.
4. Шар додавання додає початковий вхід до вихідного значення шару конволюції, тобто додається корисна інформація про зображення на попередньому рівні обробки.
5. Після застосування блоків зі зв'язками "residual" виконується макс-пулінг, який допомагає зменшити розмір зображення та сконцентрувати увагу на найважливіших деталях.
6. Потім процес повторюється знову - застосовуються блоки зі зв'язками "residual", після яких виконується макс-пулінг. В ResNet-50 цей процес повторюється ще 3 рази.
7. На останньому етапі роботи мережі виконується повний зв'язний шар, який відповідає за класифікацію зображення на певні категорії.
8. На виході мережі отримуємо набір ймовірностей, які відображають ймовірність належності зображення до кожної з категорій.

2.4 Архітектури нейромереж придатні для виконання завдання

AlexNet - це згорткова нейронна мережа, розроблена в 2012 році, яка вперше дала значну перевагу в обробці зображень з використанням глибокого навчання. Назва мережі походить від імені автора - Алекса Кріжевського.

Архітектура мережі містить 5 згорткових шарів (рисунок 2.2), з яких перші два мають велику кількість фільтрів, що допомагає виявляти прості та складні ознаки на зображеннях. Після кожного згорткового шару застосовується шар пулінгу, щоб зменшити розмір вихідного зображення та збільшити ефективність мережі.

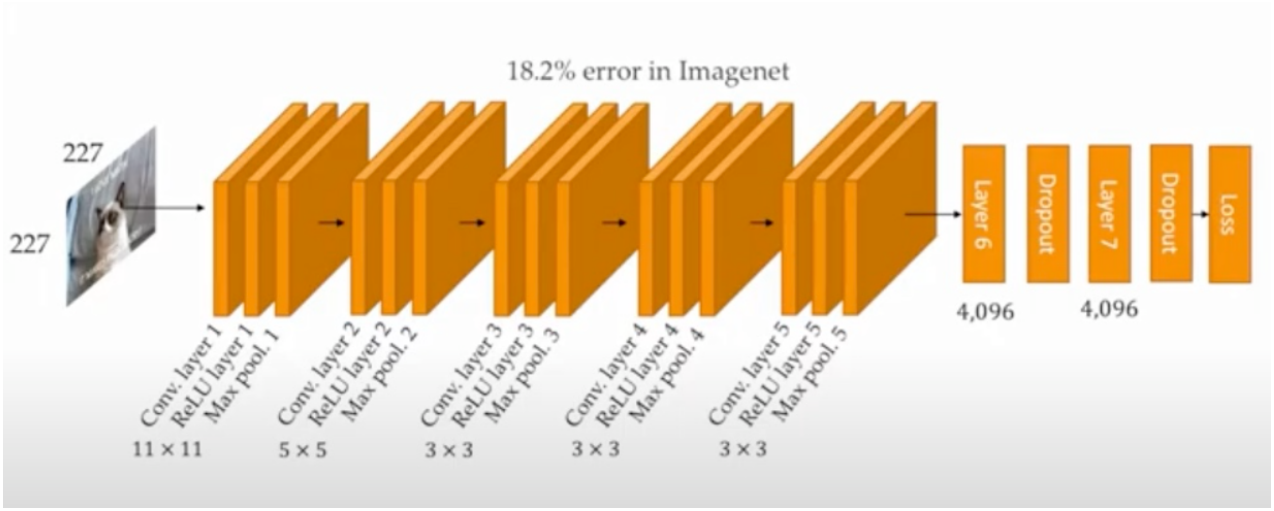


Рисунок 2.2 - Архітектура AlexNet

Після 5-го згорткового шару слідують 3 повнозв'язних шари, кожен з яких має розмірність 4096 нейронів. Ці шари виконують функцію класифікації зображення, згенерованого з попереднього згорткового шару.

Одна з ключових особливостей AlexNet - використання техніки Dropout. Ця техніка полягає в тому, що випадковим чином деякі нейрони вимикаються на кожній ітерації, що допомагає запобігти перенавчанню та підвищити ефективність мережі.

Наступним варіантом архітектури було обрано VGG, вона приховує в своїй назві групу дослідників з Університету Оксфорда, яка займається дослідженням

комп'ютерного зору та обробки зображень. У 2014 році команда VGG опублікувала статтю "Very Deep Convolutional Networks for Large-Scale Image Recognition", в якій вони запропонували архітектуру VGG для класифікації зображень.

Архітектура VGG (рисунок 2.3) складається з декількох блоків, кожен з яких складається з послідовності згорткових шарів та шарів пулінгу. Кожен блок має фіксовану кількість згорткових шарів та шарів пулінгу, що дозволяє легко налаштовувати глибину мережі. Архітектура VGG має велику кількість параметрів, що зробило її дуже потужною, але водночас вимагає великої кількості обчислювальних ресурсів для тренування та застосування.

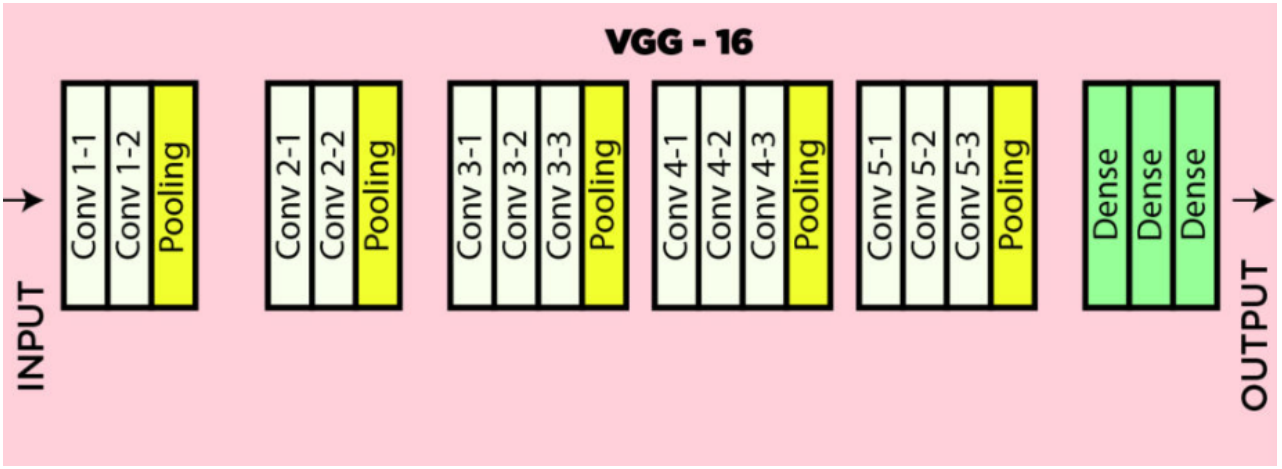


Рисунок 2.3 - Архітектура VGG16

VGG стала популярною моделлю для класифікації зображень та була використана у багатьох дослідженнях та застосуваннях, включаючи розпізнавання облич, детекцію об'єктів та інші задачі комп'ютерного зору. Також VGG була використана як базова модель для багатьох подальших розробок та модифікацій, наприклад, у VGG16 та VGG19.

VGG16 і VGG19 є глибокими нейронними мережами, обидві мережі використовуються для класифікації зображень і є основними архітектурами для цієї задачі.

Архітектура VGG16 складається з 16 шарів, включаючи 13 згорткових шарів та 3 повнозв'язних шарів. Згорткові шари складаються з фільтрів різного розміру, які виконують згортання зображень, щоб вилучити різноманітні ознаки зображень. Повнозв'язні шари використовують ці ознаки для класифікації зображень на різні категорії.

Архітектура VGG19 має таку саму структуру, як VGG16, але з 3 додатковими згортковими шарами. Це забезпечує ще більшу точність класифікації, але при цьому потребує більшої обчислювальної потужності для тренування та використання.

VGG16 та VGG19 стали популярними архітектурами для задач класифікації зображень завдяки їхній високій точності та простоті у реалізації. Вони також часто використовуються як базові моделі для багатьох інших завдань у глибинному навчанні.

Для виконання поставленої на бакалаврську кваліфікаційну роботу задачі є достатньо потужні обчислювальні здібності, то для класифікації зображень можна використовувати більш глибокі і складні архітектури, такі як ResNet і VGG, оскільки вони зазвичай дають кращі результати в порівнянні з AlexNet.

Зокрема, VGG має більшу кількість шарів, що дозволяє отримати високу точність класифікації зображень. Також ResNet, завдяки використанню блоків зі зворотним зв'язком, дозволяє побудувати ще більш глибоку мережу з меншою кількістю параметрів, що робить його популярним використовуваним варіантом для багатьох завдань в області комп'ютерного зору.

Висновки до розділу 2

У даному розділі було розглянуто три нейромережі, що були створенні для виконання завдання та проведено огляд трьох відомих архітектур глибоких нейронних мереж - ResNet, AlexNet і VGG. Кожна з цих архітектур має свої переваги та недоліки, що можуть впливати на їх ефективність у вирішенні конкретної задачі класифікації зображень.

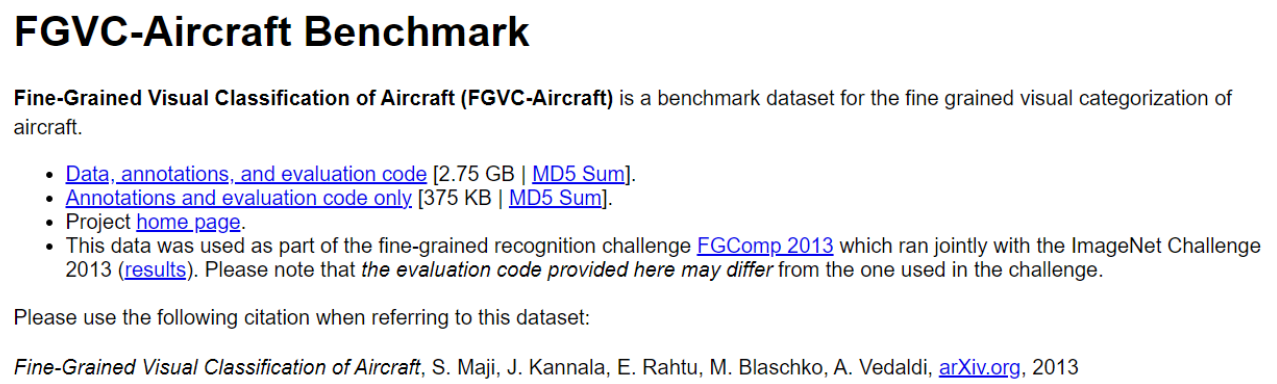
Після ретельного аналізу та порівняння результатів виконання алгоритмів класифікації зображень на різних архітектурах, було визначено, що ResNet демонструє високу точність класифікації на широкому спектрі даних зображень, що зробило його обраним варіантом для вирішення поставленої задачі класифікації зображень.

Отже, на підставі проведеного дослідження та отриманих результатів, можна зробити висновок, що архітектура ResNet є ефективним інструментом у вирішенні задач класифікації зображень, що потребують високої точності.

3 ПІДГОТОВКА ДАНИХ ТА РОЗРОБКА СТРУКТУРИ СИСТЕМИ

3.1 Опис вхідних даних та структури системи

Для виконання БКР будемо брати готовий датасет з інтернет ресурсу robots.ox.ac.uk, на якому вже є готовий датасет з фотографіями більш ніж 50 різних типів літаків. На рисунках 3.1 та 3.2 зображено інтерфейс інтернет ресурсу з датасетом та вміст датасету відповідно.



FGVC-Aircraft Benchmark

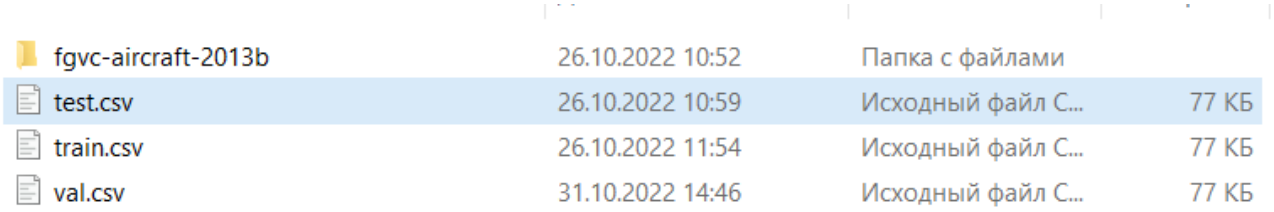
Fine-Grained Visual Classification of Aircraft (FGVC-Aircraft) is a benchmark dataset for the fine grained visual categorization of aircraft.

- Data, annotations, and evaluation code [2.75 GB | MD5 Sum].
- Annotations and evaluation code only [375 KB | MD5 Sum].
- Project home page.
- This data was used as part of the fine-grained recognition challenge FGComp 2013 which ran jointly with the ImageNet Challenge 2013 (results). Please note that *the evaluation code provided here may differ* from the one used in the challenge.

Please use the following citation when referring to this dataset:

Fine-Grained Visual Classification of Aircraft, S. Maji, J. Kannala, E. Rahtu, M. Blaschko, A. Vedaldi, arXiv.org, 2013

Рисунок 3.1 - Сторінка для завантаження датасету фото літаків



fgvc-aircraft-2013b	26.10.2022 10:52	Папка с файлами	
test.csv	26.10.2022 10:59	Исходный файл C...	77 КБ
train.csv	26.10.2022 11:54	Исходный файл C...	77 КБ
val.csv	31.10.2022 14:46	Исходный файл C...	77 КБ

Рисунок 3.2 - Дані з архіву

Але в базовому датасеті всі фото літаків складені в одній папці (рисунок 3.3), що робить дані непридатними для роботи.

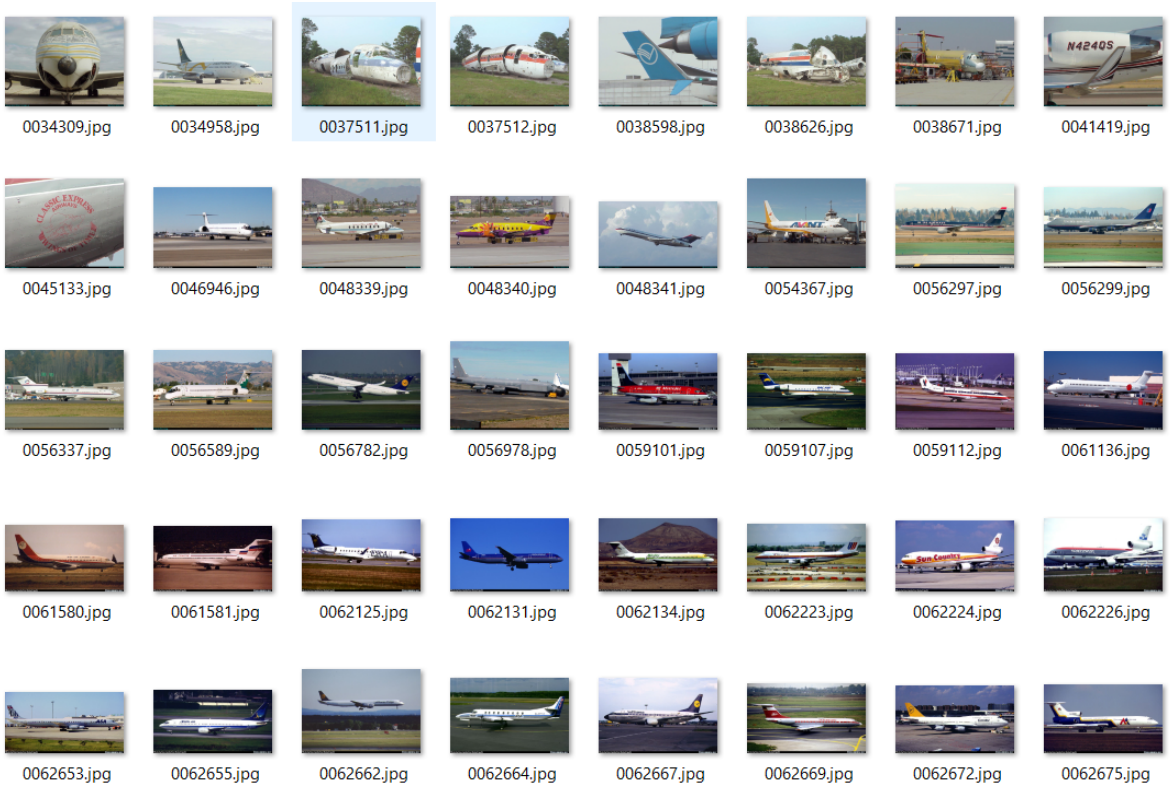


Рисунок 3.3 - Всі фото літаків водній папці

Як відомо для правильної роботи системи потрібно розсортувати всі фото по папкам, кожна папка повинна мати назву того виду літака фото котрого будуть складатись в цю папку. Автори датасету залишили в ньому текстовий файл (рисунок 3.4), в якому можна переглянути, який тип літака зображений на фото.

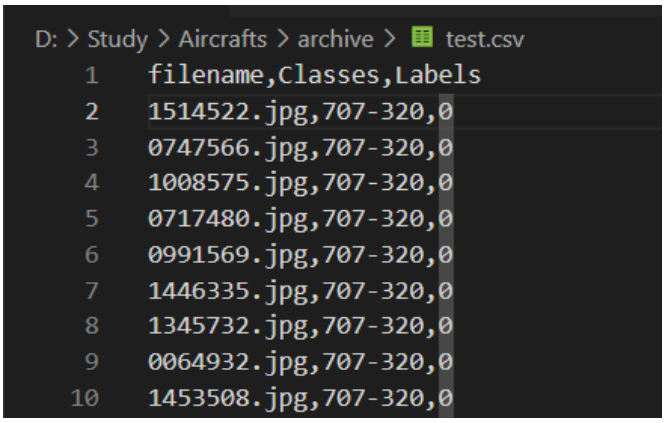


Рисунок 3.4 - Вигляд файлу з іменами фото та типом літака котрий на ньому зображено

Для сортування нашого датасету створимо консольний додаток котрий створить потрібні папки з назвами потрібних нам видів літаків та розклав фото по цих папках. Спершу створимо клас котрий описуватиме дані з файлу, котрий містить назву фото, тип літака та його номер (рисунок 3.5).

```
public class AirData
{
    Ссылок: 5
    public string ImageName { get; set; }

    Ссылок: 4
    public string Name { get; set; }

    Ссылок: 1
    public string Number { get; set; }
}
```

Рисунок 3.5 - Клас для опису даних про літаки

Далі створимо 3 змінних типу string для збереження таких шляхів як: до файлу з записаними назвами та типами, до папки з фото та до папки в котрій будуть створюватись папки з розсортованими фото, а також створили колекцію для збереження зчитаних з файлу даних (рисунок 3.6).

```
string[] lines = System.IO.File.ReadAllLines(@"D:\Study\Aircrafts\archive\val.csv");
string src = @"D:\Study\Aircrafts\archive\fgvc-aircraft-2013b\fgvc-aircraft-2013b\data\images\";
string dest = @"C:\Users\acsel\source\repos\SuChecker\Aircrafts\";

List<AirData> air = new List<AirData>();
```

Рисунок 3.6 - Збереження потрібних шляхів до папок та файлів

Потім з читавши всі рядки в масив, пройдемося по них циклом та сформуємо дані для колекції (рисунок 3.7).

```
foreach (string line in lines)
{
    string[] splitted = line.Split(',');
    AirData airData = new AirData()
    {
        ImageName = splitted[0],
        Name = splitted[1],
        Number = splitted[2]
    };

    air.Add(airData);
}
```

Рисунок 3.7 - Цикл для формування колекції

Останнім пунктом в створені скрипту для сортування фото буде створення ще одного циклу для проходження по підготовленим даним колекції та вибираючи назви літаків копіювати їх в створені папки з їх назвами (рисунок 3.8).

```
foreach (AirData airData in air)
{
    string path = @"C:\Users\acsel\source\repos\SuChecker\Aircrafts\" + airData.Name;
    if (Directory.Exists(path) != true)
    {
        Directory.CreateDirectory(path);
        System.IO.File.Copy(src + airData.ImageName, dest + airData.Name + "\\" + airData.ImageName);
    }
    else
    {
        System.IO.File.Copy(src + airData.ImageName, dest + airData.Name + "\\" + airData.ImageName);
    }
}
```

Рисунок 3.8 - Цикл копіювання фото

Запустимо наш скрипт та почекаємо поки скрипт відпрацює та в папці, що ми відвели під створення папок з фото літаків з’являться папки зі скопійованими фото (рисунок 3.9).

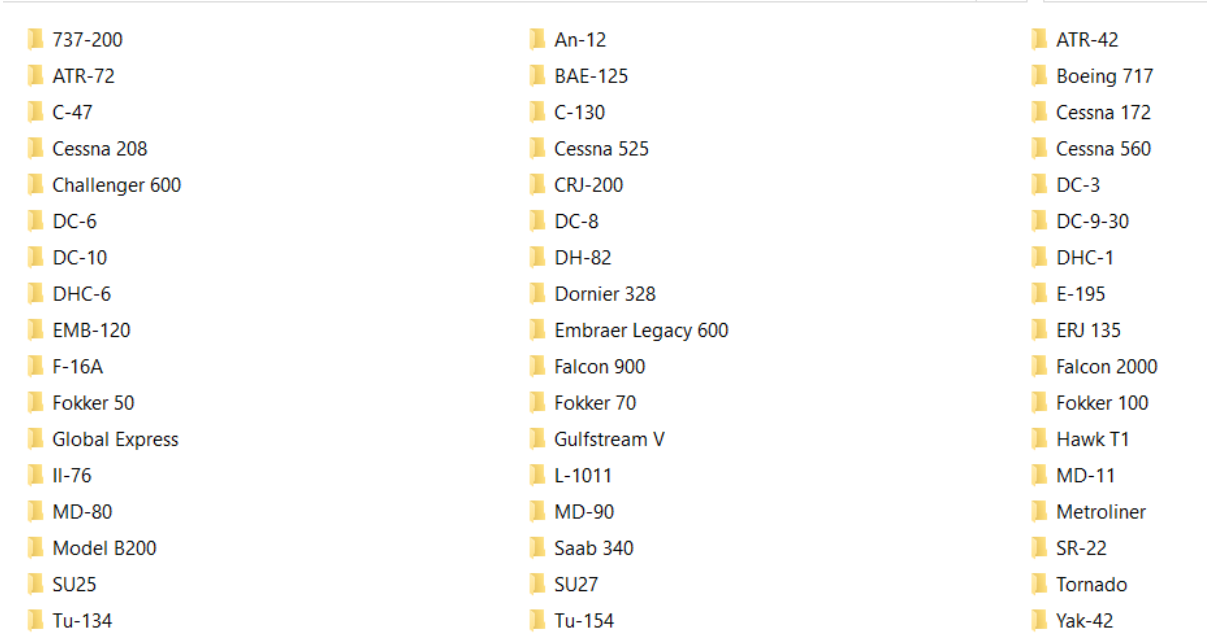


Рисунок 3.9 - Розсортовані фото літаків по папкам

Відкривши будь-яку папку ми зіткнемося з проблемою, а саме з тим, що для успішного проведення навчання нейромережі для класифікації фото, треба більше 100 фото для кожного літака, але реальність така, що в датасеті було по 60 фото на кожен тип літака.

Тож для роботи над завданнями було вирішено обрати 4 типи літаків та розширити вже наявний перелік фото, отримані з датасету, фотографіями з інтернету, для доведення колекції кожного літака до більш ніж 200 фото для кожного типу. Для виконання завдання візьмемо 4 типи літаків такі як: Ан-12, ATR-42, Spitfire та Cessna 208, ці типи літаків і отримали збільшення кількості фото в папках.

В свою чергу, папки, що містили в собі фото цих літаків були поділені ще на 2, одна для навчання друга для тестування нейромереж (рисунок 3.10).

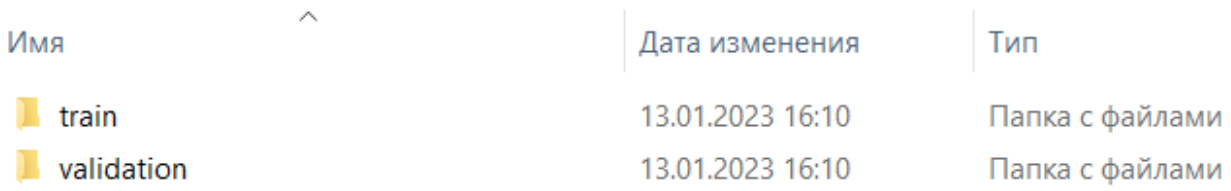


Рисунок 3.10 - Розділені на 2 категорії фото літаків

Фото між папками для тренування та тестування діляться таким чином, що 80% всих фото літака знаходяться в папці для тренування, а інші 20% в папці для тестування, в середині папок знаходяться папки з назвами обраних літаків та з їхніми фото.

Для успішного навчання нейромережі завданням якої є класифікація фото, фотографії повинні бути високої якості та розмірності, щоб забезпечити достатню кількість деталей для аналізу нейромережею. Крім того, фотографії повинні бути репрезентативними для задачі класифікації, тобто містити достатньо візуальних ознак, що дозволяють розрізняти об'єкти різних класів.

Також, для забезпечення успішного навчання, важливо мати достатню кількість фотографій для кожного класу. Ідеально, якщо кількість фотографій в кожному класі буде однаковою, але в реальних умовах це може бути складно досягнути. В нашому випадку вдалось зібрати датасет фото, для навчання нейромережі, розміром в 1000 фото, або ж 250 фото для кожного з 4 типів літаків.

3.2 Проектування розробки вебзастосунку для роботи з клієнтами

При проектуванні вебзастосунку, перш за все, необхідно визначити цілі та потреби, які він має задовольняти. Далі необхідно визначити функціональні вимоги до вебзастосунку і його основні характеристики, такі як обсяг бази даних, потреба в зберіганні файлів, способи взаємодії з користувачами та інші.

Також важливим етапом проектування є розробка інтерфейсу користувача. Інтерфейс повинен бути зрозумілим та зручним у використанні, мати логічну

структуру та відповідати потребам цільової аудиторії. Під час розробки інтерфейсу важливо враховувати не тільки естетичні аспекти, а й принципи юзабіліті та доступності.

Після розробки концепції та планування функціональності і інтерфейсу, необхідно визначити технічні вимоги до вебзастосунку, такі як вимоги до мов програмування, баз даних, вебсерверів тощо [18].

Розробка вебзастосунку - це процес створення програмного забезпечення, яке можна використовувати через Інтернет за допомогою веббраузера. Веб-застосунки можуть бути створені для різних цілей, таких як електронна комерція, онлайн-банкінг, соціальні мережі, онлайн-ігри та багато іншого.

Основні компоненти вебзастосунку включають вебсервер, базу даних та клієнтський інтерфейс. Веб-сервер забезпечує зв'язок між клієнтом та сервером і оброблює запити, які надходять від клієнтів. База даних зберігає інформацію, яка використовується вебзастосунком. Клієнтський інтерфейс представляє собою спосіб взаємодії користувача з вебзастосунком [13].

Розробка вебзастосунку включає в себе такі етапи, як проектування, розробка, тестування та розгортання. Проектування включає в себе створення макетів інтерфейсу користувача та визначення функціональності вебзастосунку. Розробка включає в себе програмування серверної та клієнтської частин вебзастосунку. Тестування включає в себе перевірку функціональності, безпеки та продуктивності вебзастосунку. Розгортання означає встановлення вебзастосунку на сервері та налаштування його для використання в Інтернеті.

Структура системи, яка складається з вебсайту, що розпізнає об'єкт на фото за 3 нейронними мережами, описана нижче [12-15].

1. Клієнт (користувач) відправляє запит на вебсайт і вказує зображення, яке потрібно класифікувати.

2. Серверний апарат вебсайту отримує запит і передає його до сервера, на якому працюють нейромережі.

3. На сервері запускаються нейромережі і проводиться класифікація зображення за кожною з них.

4. Результати класифікації повертаються на сервер вебсайту.

5. Веб-сайт обробляє результати класифікації і повертає їх клієнту.

6. Клієнт отримує результати класифікації на вебсайті.

Така структура системи може використовуватись в різних сферах, наприклад, для класифікації продуктів на фотографіях в інтернет-магазині, або для розпізнавання облич людей на фотографіях у системі безпеки.

Створення застосунку, що розпізнає об'єкти на фото за допомогою нейромережі може бути складним процесом. Ми вже зібрали датасет з фото, на яких зображені об'єкти, які ми хочемо розпізнавати. Потім, буде виконано процес створення та навчання нейромереж, в нашому випадку це ResNet-50 та ResNet-101.

Для створення застосунку можна використовувати різні інструменти та технології. Для розробки вебсайту використано мови програмування, такі як C#, JavaScript, HTML, та CSS. Для забезпечення роботи зображень, використано бібліотеки з обробки зображень, такі як OpenCV. Сам вебзастосунок буде реалізований на основі технології ASP.Net MVC [16].

Окремі компоненти системи можна збирати окремо, тестувати та поєднувати. Можна розробити вебсайт з функцією завантаження фотографій та використання навченої нейромережі для розпізнавання об'єктів на цих фотографіях. Можна також використовувати мережу API для забезпечення зв'язку між різними компонентами системи.

Якщо говорити детальніше, то ASP.NET MVC - це фр еймворк для розробки вебдодатків на мові програмування C#, який використовується для створення

додатків з відокремленою логікою, що сприяє полегшенню розробки, тестування та розгортання [12].

Фреймворк дозволяє відокремити логіку бізнес-процесів від логіки відображення (представлення) на стороні клієнта, а також відокремити ці дві логіки від логіки роботи з моделю (рисунок 3.11). Це дозволяє розробникам легко змінювати різні частини додатку, не впливаючи на решту коду.

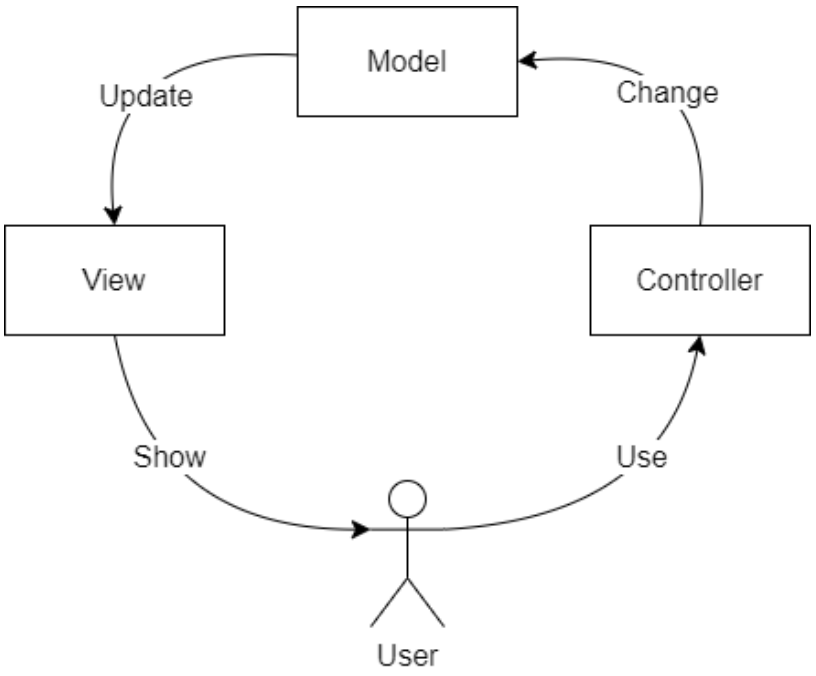


Рисунок 3.11 - Схема роботи ASP.Net MVC з користувачем

ASP.NET MVC також надає широкі можливості для налаштування та розширення функціоналу додатку, а також дозволяє використовувати різні пакети та бібліотеки для розширення можливостей. В цілому, ASP.NET MVC є потужним інструментом для розробки вебдодатків, який дозволяє розробникам швидко та ефективно створювати високоякісні додатки з відокремленою логікою та широким функціоналом [13,17].

Створення вебзастосунку на ASP.NET MVC розділимо на кілька етапів.

1. Налаштування робочого середовища:

- встановлення Visual Studio;
- встановлення пакетів розширення ASP.NET та .NET Framework.

2. Створення нового проекту ASP.NET MVC:
   - вибір шаблону "ASP.NET Web Application";
   - вибір типу проекту "MVC";
   - встановлення назви та місця розташування проекту.
3. Налаштування проекту:
   - додавання моделей даних;
   - створення контролерів;
   - налаштування маршрутів;
   - додавання ресурсів (зображень, стилів, скриптів).
4. Розробка сторінок та їх логіки:
   - створення HTML-шаблонів сторінок;
   - реалізація логіки в контролерах;
   - взаємодія з базою даних.
5. Тестування та налагодження:
   - перевірка роботи сторінок та їх логіки;
   - виправлення помилок та багів;
   - підготовка проекту до релізу.

Додавання моделей даних - це процес створення та додавання об'єктів даних, які використовуються в програмі, у вигляді моделей даних. Моделі даних описують структуру та властивості даних, що зберігаються в базі даних або файловій системі. Цей процес зазвичай включає в себе визначення полів моделі даних, типів даних, залежностей між моделями та їхніми взаємозв'язками [19].

Створення контролерів - це процес створення об'єктів програмного коду, які взаємодіють з даними та виконують логіку програми. Контролери зазвичай відповідають за обробку запитів користувачів та взаємодію з моделями даних. Цей

процес зазвичай включає в себе створення функцій-контролерів, їх обробку даних, тестування та оптимізацію коду.

Налаштування маршрутів - це процес визначення шляхів, за якими користувач може отримати доступ до різних частин програми. Маршрути зазвичай включають в себе URL-адресу та відповідний контролер, який буде обробляти запит. Цей процес зазвичай включає в себе визначення маршрутів для різних дій, включаючи відображення статичних сторінок, відправку даних з форм, взаємодію з базою даних та інші операції [11-13].

Реалізація логіки в контролерах - це процес створення програмного коду, який виконується при запиті користувача та взаємодії з даними. Цей процес зазвичай включає в себе написання функцій-контролерів, які отримують запит від користувача та виконують певну логіку програми, таку як отримання даних з бази даних, відображення сторінок, обробка даних, валідація введених даних та інші операції.

Перевірка роботи сторінок та їх логіки - це процес перевірки правильності функціонування програми, включаючи відображення сторінок та виконання логіки програми. Цей процес зазвичай включає в себе тестування кожної сторінки та функціональної можливості, щоб переконатися, що програма працює вірно та не містить помилок. Для цього можуть використовуватися різноманітні інструменти, такі як модульні тести, функціональні тести, інструменти відладки та інші. В результаті цього процесу можна виявити та виправити будь-які помилки в програмі, що забезпечить правильне та надійне її функціонування.

Паралельно зі створенням вебзастосунку, для того аби не втрачати час, буде почато процес створення та навчання нейромереж. Процес створення нейромереж ResNet для класифікації фото на мові програмування Python може бути розбитий на наступні етапи:

– завантажити необхідні бібліотеки для роботи з Python, такі як TensorFlow та Keras;

– побудувати та натренувати нейромережу ResNet-50 за допомогою Keras. Для цього необхідно використати вже наявну реалізацію ResNet-50 з бібліотеки Keras та натренувати її на нашому наборі даних;

– оцінити ефективність натренованої моделі на тестовому наборі даних;

– зберегти натреновану модель для подальшого використання;

– використовуючи збережену модель, можна створити застосунок, який буде класифікувати фотографії на основі натренованої нейромережі.

У процесі розробки такого застосунку важливо враховувати налаштування таких як кількість шарів та їх розмір, швидкість навчання тощо, щоб досягти максимальної точності та ефективності моделі.

Висновки до розділу 3

У даному розділі було описано важливі аспекти підготовки даних для навчання нейромережі, включаючи вибір і обробку фото, розмітку даних та валідацію навчального набору. Було також розглянуто основні принципи роботи ASP.NET та процес розробки вебзастосунку на даній платформі.

Підготовка даних є ключовим етапом при розробці нейромережі, оскільки від якості та кількості даних залежить точність та ефективність навчання моделі. Описані методи та інструменти дозволяють ретельно підготувати дані до роботи з ними.

Платформа ASP.NET є потужним інструментом для розробки вебзастосунків з високим рівнем безпеки та масштабованості. Розглянуті етапи розробки вебзастосунку на ASP.NET дозволяють створити ефективний та функціональний продукт для класифікації фото з використанням нейромережі.

4 СТВОРЕННЯ НЕЙРОМЕРЕЖ ТА ВЕБСАЙТУ ДЛЯ РОБОТИ З НЕЙРОМЕРЕЖАМИ

4.1 Створення нейромережі ResNet-50 з використанням ML.Net

Спочатку попрацюємо з моделлю ML.Net, відкриємо Visual Studio 2022 та створимо вебзастосунок на якому в подальшому будемо тестувати створені нами нейромережі. До створеного вебзастосунку додамо модель машинного навчання та виберемо в ній сценарій класифікації зображень (рисунок 4.1).

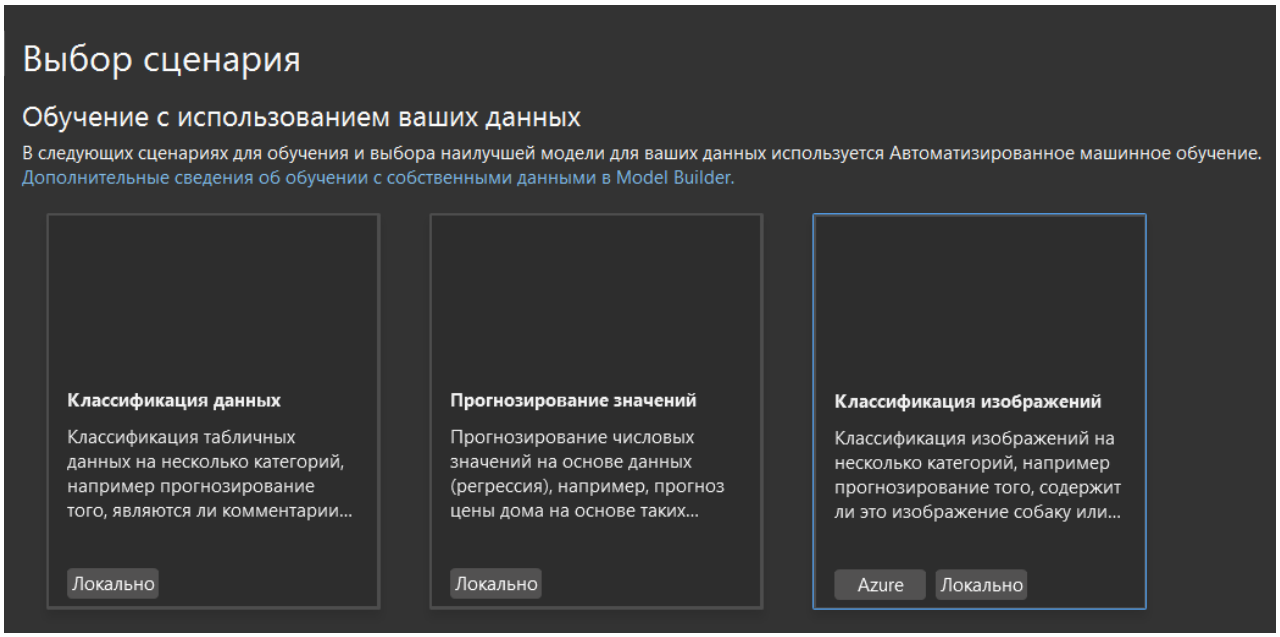


Рисунок 4.1 – Інтерфейс вибору сценарію навчання

Наступним кроком виберемо середовище навчання нашої моделі класифікації зображень, в нашому випадку оберемо локальне навчання моделі на наявному ПК (рисунок 4.2).

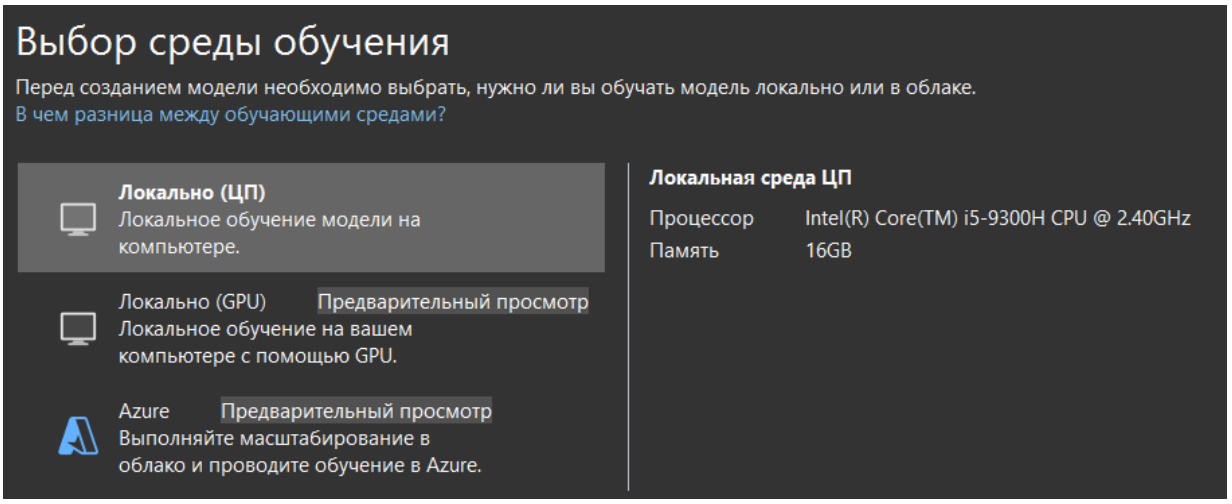


Рисунок 4.2 – Інтерфейс вибору середовища навчання

Наступним кроком буде вибір даних для навчання. Всі дані повинні бути розділені по папках, котрі будуть підписані назвою об’єкта зображеного на фото (рисунок 4.3).

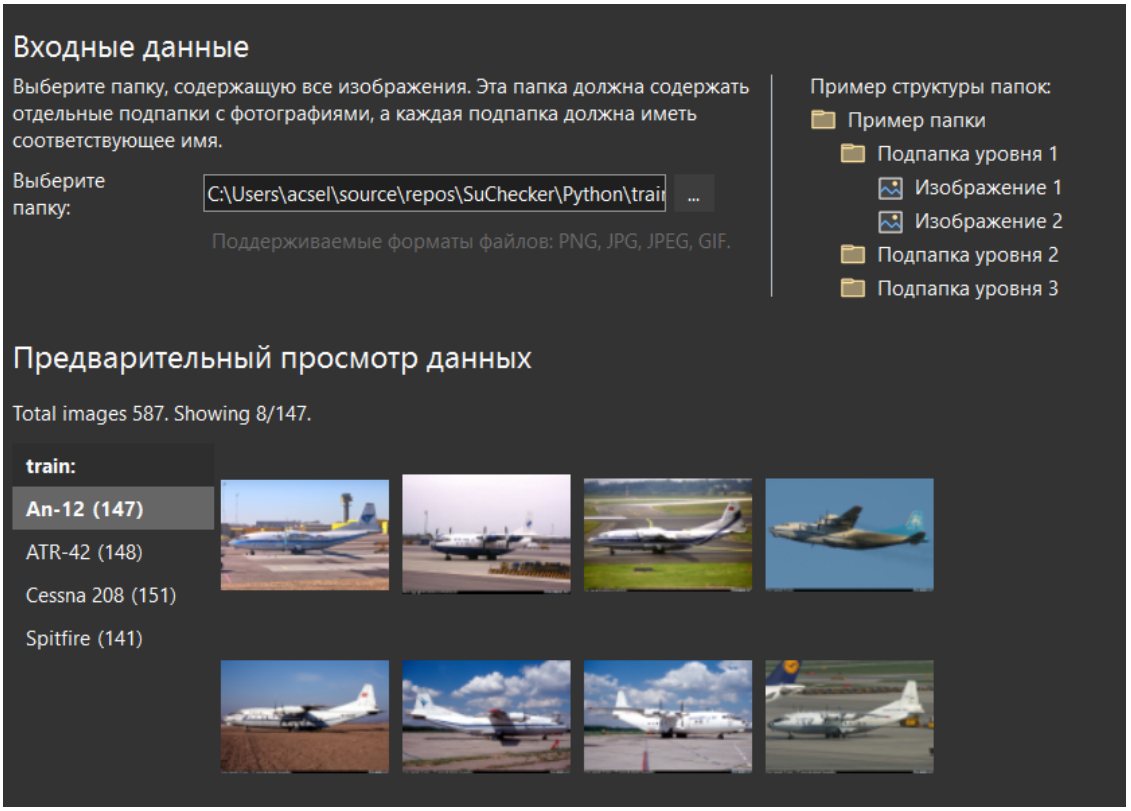


Рисунок 4.3 – Інтерфейс вибору даних для навчання

Наступний крок найважливіший, навчання моделі. З цим завданням нейромережа справилась дуже добре, провівши навчання з точністю в 92.16%, що є дуже добрим показником (рисунок 4.4).

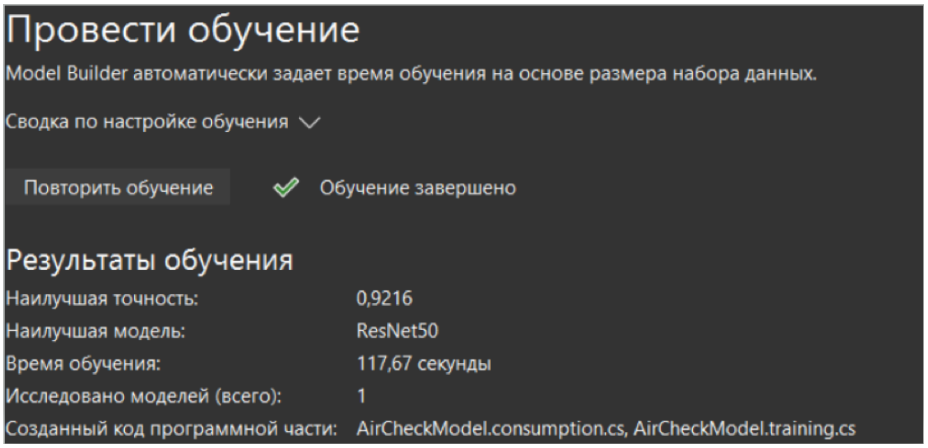


Рисунок 4.4 – Завершене навчання моделі

Наостанок перевіримо модель на коректність класифікації зображень та дамо класифікувати їй зображення для тесту. В результаті модель дала правильну відповідь з точністю в 89% (рисунок 4.5).

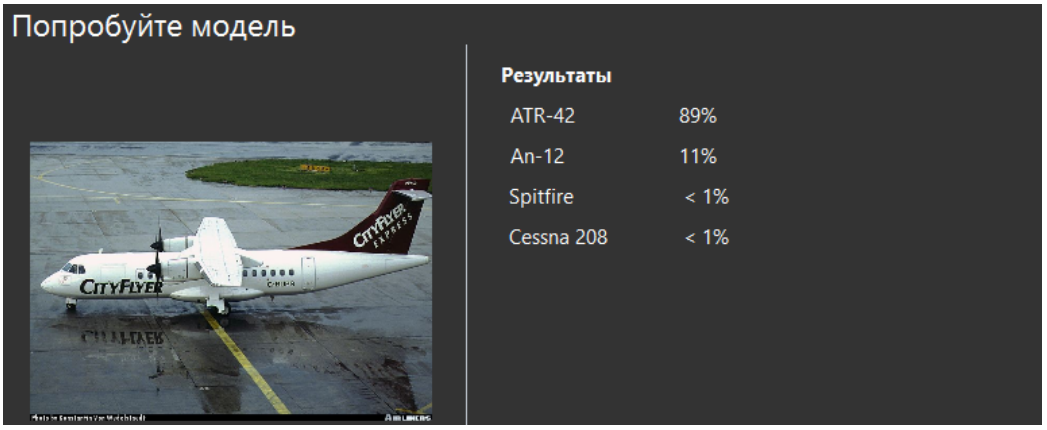


Рисунок 4.5 – Перевірка моделі

4.2 Створення нейромережі ResNet-50 та з використанням Python

Тепер створимо нейромережу з використанням мови програмування Python, а саме ResNet-50. В першу чергу імпортуємо потрібні нам бібліотеки (рисунок 4.6).

```
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
from keras.models import Model
```

Рисунок 4.6 – Імпорт бібліотек для створення

Далі створимо нашу модель ResNet-50 та додамо до неї шари Dense та Flatten. Dense шари використовуються для вивчення нелінійних зв’язків між входами та виходами, тоді як шари Flatten використовуються для перетворення багатовимірних тензорів в одновимірні вектори (рисунок 4.7).

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

# Freeze the layers in the pre-trained model
for layer in resnet.layers:
    layer.trainable = False

# Add a new fully connected layer
x = Flatten()(resnet.output)
x = Dense(256, activation='relu')(x)
x = Dense(128, activation='relu')(x)
prediction = Dense(4, activation='softmax')(x)
```

Рисунок 4.7 – Створення моделі ResNet-50

Далі створимо та скомпілюємо головну модель для проведення навчання та задамо їй потрібні параметри (рисунок 4.8).

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

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

Рисунок 4.8 – Створення моделі навчання нейромережі

Наступним кроком буде витягування потрібних нам для навчання зображень літаків з допомогою ImageGenerator (рисунок 4.9).

```
# 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)

train_Path = 'C:/Users/acsel/source/repos/SuChecker/Python/train/'
test_Path = 'C:/Users/acsel/source/repos/SuChecker/Python/validation/'

train_generator = train_datagen.flow_from_directory(train_Path,
                                                    target_size=(224, 224),
                                                    batch_size=32,
                                                    class_mode='categorical')
test_generator = test_datagen.flow_from_directory(test_Path,
                                                  target_size=(224, 224),
                                                  batch_size=32,
                                                  class_mode='categorical')
```

Рисунок 4.9 – Витягування зображень для навчання моделі

Ну і найголовніше - це сам процес навчання, для цього виконаємо методи fit_generator та evaluate_generator, передавши їх параметри навчання та зображення (рисунок 4.10).

```
# 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 4.10 – Виклик методів для навчання моделі

Наостанок дамо нашій створеній моделі можливість провести класифікацію літака Cessna 208 та виведемо назву класифікованого літака в консоль (рисунок 4.11).

```
# Load the image to classify
img_path = 'C:/Users/acsel/source/repos/SuChecker/Python/train/Cessna 208/62659_1648840269.jpg'
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 = ['An-12', 'ATR-42', 'Cessna 208', 'Spitfire']
predicted_class_index = np.argmax(predictions)
predicted_class_name = class_names[predicted_class_index]
print("The image is a", predicted_class_name)
```

Рисунок 4.11 – Проведення перевірки навченої моделі на спроможність класифікувати зображення

Після написання коду програми запустимо її та отримаємо в консолі кількість фото котрі ідуть на тренування та котрі йдуть на перевірку, також

побачимо прогрес навчання котрий триватиме 40 ітерацій, на кожну з яких буде в середньому витрачено 73 секунди та найголовніше буде отримано точність в 92% і на кінець буде виведено результат класифікації літака на зображенні (рисунок 4.12).

```
PS D:\Study\Python>  & 'C:\Python311\python.exe' '--' 'D:\Study\Python\NeuralImage\resnet50_PC.py'
Found 587 images belonging to 4 classes.
Found 215 images belonging to 4 classes.
Epoch 40/40
19/19 [==============================] - 73s 4s/step - loss: 0.7328 - accuracy: 0.9223
1/1 [==============================] - 2s 2s/step
The image is a Cessna 208
PS D:\Study\Python>
```

Рисунок 4.12 – Результат навчання моделі ResNet-50

Після навчання ResNet-50, перейдемо до створення ResNet-101. Код для створення та навчання моделі ResNet-101 буде взято з другої нейромережі, єдиною різницею між ними буде додавання замість моделі ResNet-50, модель ResNet-101 (рисунок 4.13).

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

Рисунок 4.13 – Додавання до моделі навчання архітектуру ResNet101

Після написання коду програми запустимо її та отримаємо в консолі кількість фото котрі ідуть на тренування та котрі йдуть на перевірку, також побачимо прогрес навчання котрий триватиме 40 ітерацій, на кожну з яких буде в середньому витрачено 104 секунди та найголовніше буде отримано точність в 94% і на кінець буде виведено результат класифікації літака на зображенні (рисунок 4.14).

```
Found 587 images belonging to 4 classes.
Found 215 images belonging to 4 classes.
Epoch 40/40
19/19 [==============================] - 104s 5s/step - loss: 0.6838 - accuracy: 0.9435
1/1 [==============================] - 3s 3s/step
The image is a Cessna 208
```

Рисунок 4.14 – Результат навчання моделі ResNet101

В результаті дослідження зроблено висновок, що ResNet-101 навчається довше ніж ResNet-50, хоча кількість епох та потужності ПЗ однакові. Це зумовлено тим, що ResNet-101 має більшу кількість шарів (101) ніж ResNet-50 (50). Це означає, що ResNet-101 має більше параметрів, які потрібно навчити, що зазвичай призводить до більшого часу навчання.

4.3 Створення зовнішнього сценарію для запуску нейромереж створених на Python з середовища C#

Після створення всіх потрібних нейромереж перейдемо до створення методу для надання можливості одночасно давати запити всім 3 нейромережам для класифікації літаків на зображенні.

Спершу на мові програмування Python зробимо скрипт зовнішнього сценарію для виклику класифікації літака на зображенні з середовища .Net, отже створимо метод для класифікації котрий прийматиме потрібну нам модель нейромережі та назву фото.

```python
def prediction(model, path):
    img = image.load_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 = ['An-12', 'ATR-42', 'Cessna 208', 'Spitfire']
    predicted_class_index = np.argmax(predictions)
    predicted_class_name = class_names[predicted_class_index]
    return predicted_class_name
```

Рисунок 4.15 – Метод для класифікації літаків

Далі створимо змінні в котрі зберігатимемо параметри для класифікації, а саме назву нейромережі та назву фото. Їх ми будемо передавати як параметри при виклику скрипта з коду C# (рисунок 4.16).

```python
type_ResNet = sys.argv[1]
name = sys.argv[2]

fullPath = r"C:\\Users\\acsel\\source\\repos\\BachelorWork\\BachelorWork\\Pictures\\" + name;
CATEGORIES = ['An-12', 'ATR-42', 'Cessna 208', 'Spitfire']

IMAGE_SIZE = [224, 224]
```

Рисунок 4.16 – Збереження потрібних змінних

Наостанок напишемо код для визначення потрібної нейромережі для класифікації літака на зображенні (рисунок 4.17).

```
if type_ResNet == "ResNet50":
    model = load_model('D:/Study/Python/resnet50_chat.h5')
    predicted_class_name = prediction(model, fullPath)
    print(predicted_class_name)
if type_ResNet == "ResNet101":
    model = load_model('D:/Study/Python/resnet101_chat.h5')
    predicted_class_name = prediction(model, fullPath)
    print(predicted_class_name)
```

Рисунок 4.17 – Код для класифікації потрібної нейромережі

Далі створимо в нашому вебзастосунку метод для запуску скрипта на мові програмування Python для класифікації літака на зображені на нейромережах ResNet-50 та ResNet-101 (рисунок 4.18).

```
Ссылок: 2
public string PredictionPython(string name, string type = "ResNet50")
{
    var psi = new ProcessStartInfo();
    psi.FileName = "C:\\Python311\\python.exe";

    psi.Arguments = $"D:\\Study\\Python\\NeuralImage\\main.py {type} {name}";

    psi.UseShellExecute = false;
    psi.CreateNoWindow = true;
    psi.RedirectStandardOutput = true;
    psi.RedirectStandardError = true;

    var errors = "";
    var results = "";

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

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

    return res;
}
```

Рисунок 4.18 – Метод виклику зовнішнього сценарію Python з середовища C#

4.4 Створення вебзастосунку

Створимо застосунок з використанням фреймворку ASP.Net MVC та переглянемо його структуру (рисунок 4.19).

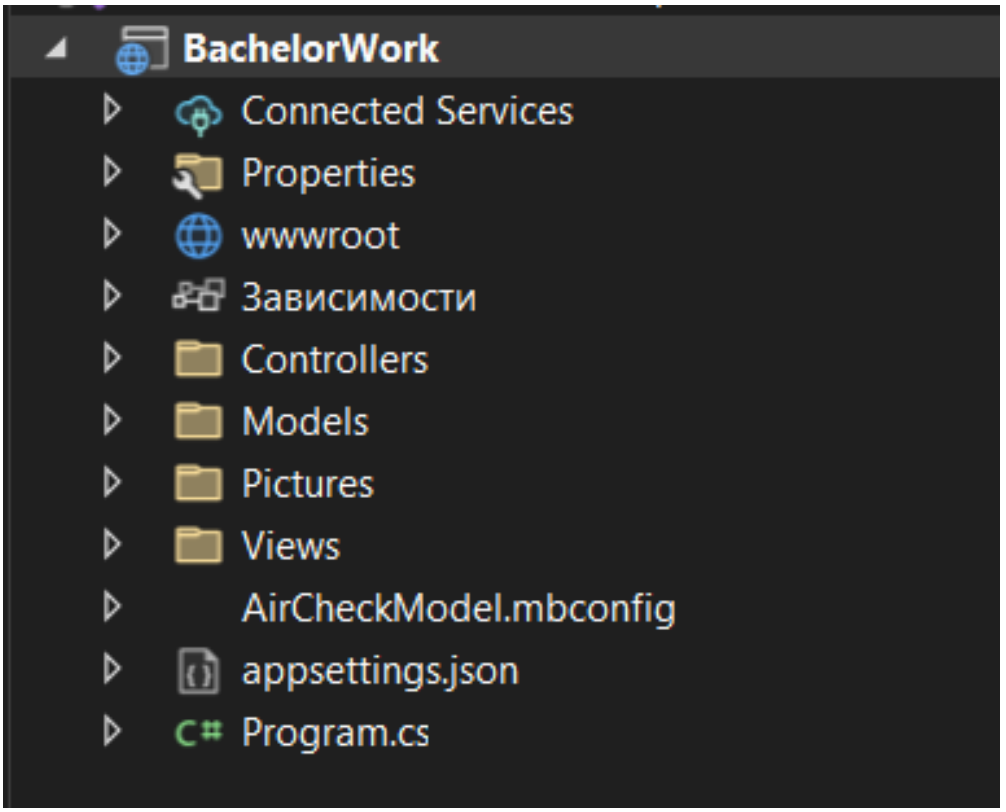


Рисунок 4.19 – Структура вебзастосунку

Створимо в нашому застосунку папку Pictures в котру будуть зберігатись всі фото, котрі будуть перевірятись користувачами.

Далі створимо метод Prediction (рисунок 4.20) котрий буде запускати класифікацію фото моделею ML.Net та робитиме 2 зовнішні сценарії для Python, результати всих 3 нейромереж повертатиме на сторінку в форматі JSON.

Спершу завантажене фото буде зберігатись до папки Pictures, а її назва буде відправлятись як параметр для зовнішніх сценаріїв та для ML.Net моделі.

```
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 mlnet = PredictMLNet(fileName);

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

Рисунок 4.20 – Метод Prediction

Для того, щоб код був більш читабельний, процес запуску зовнішніх сценаріїв було винесено в інший метод. PredictionPython (рисунок 4.21) отримує назву фото для класифікації та назву нейромережі, що повинна буде класифікувати її. Повертає метод назву літака, котрий нейромережа класифікувала на фото. В методі за допомогою класу ProcessStartInfo запускається інтерпритатор коду Python та викликається метод main.py, в котрому і проводиться класифікація.

```
public string PredictionPython(string name, string type = "ResNet50")
{
    var psi = new ProcessStartInfo();
    psi.FileName = "C:\\Python311\\python.exe";
    psi.Arguments = $"D:\\Study\\Python\\NeuralImage\\main.py {type} {name}";
    psi.UseShellExecute = false;
    psi.CreateNoWindow = true;
    psi.RedirectStandardOutput = true;
    psi.RedirectStandardError = true;
    var errors = "";
    var results = "";
    using(var process = Process.Start(psi)){
        errors = process.StandardError.ReadToEnd();
        results = process.StandardOutput.ReadToEnd();
    }
    string res = "";
    try{
        res = results.Split('\n')[3];
    } catch(Exception ex){
        res = "ERROR";
    }

    return res;
}
```

Рисунок 4.21 – Метод PredictionPython

Ну і на останок створимо інтерфейс вебзастосунку для того аби користувачі могли завантажити своє фото та отримали результати на сторінку (рисунок 4.22).

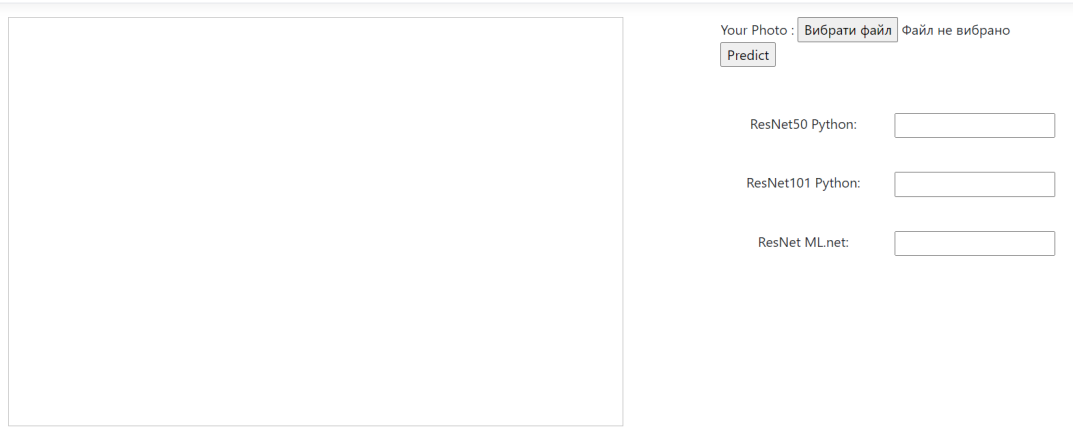


Рисунок 4.22 – Інтерфейс вебзастосунку

4.5 Тестування нейромереж та вебзастосунку

Проведемо додатково перевірки на фото літаків, на наших нейромережах, що створені на мові програмування Python (рис. 4.23-4.30).

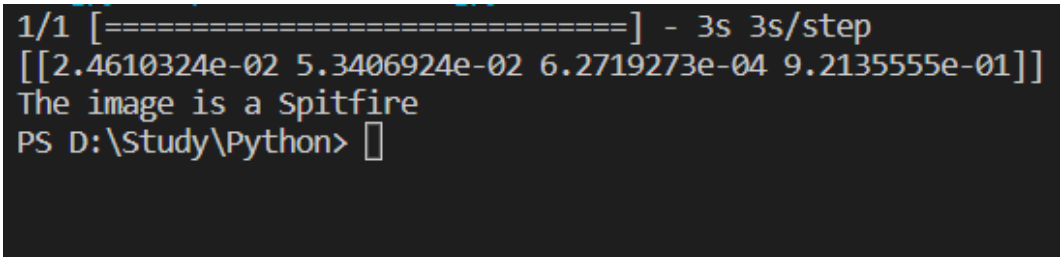


Рисунок 4.23 – Результат класифікації літака Spitfire

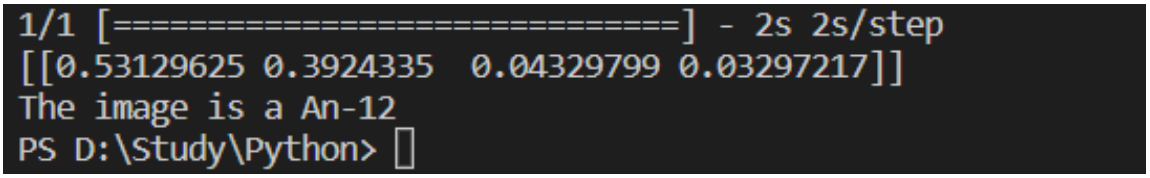


Рисунок 4.24 – Результат класифікації літака Ан-12

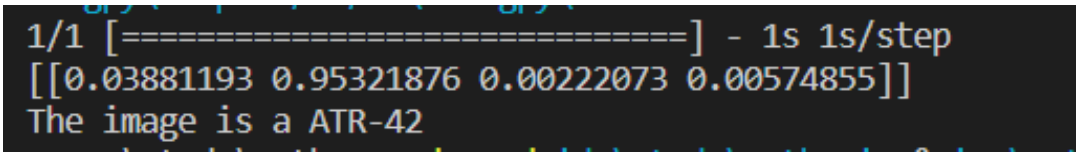


Рисунок 4.25 – Результат класифікації літака ATR-42

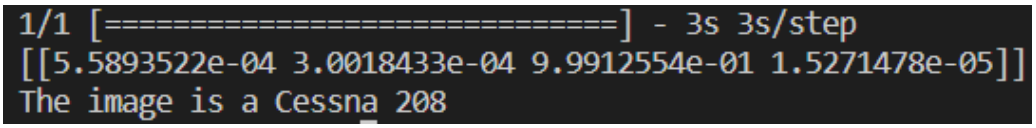


Рисунок 4.26 – Результат класифікації літака Cessna 208

На останок використаємо той самий код, але завантажимо до нього нейромережу ResNet-50 та переглянемо його результат класифікації літаків.

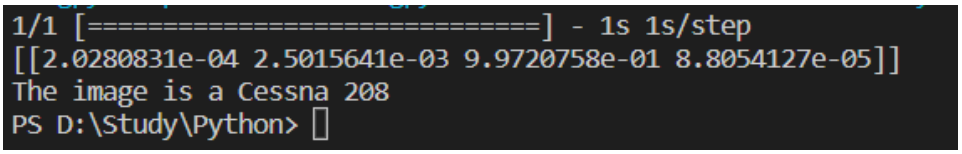


Рисунок 4.27 – Результат класифікації літака Cessna 208

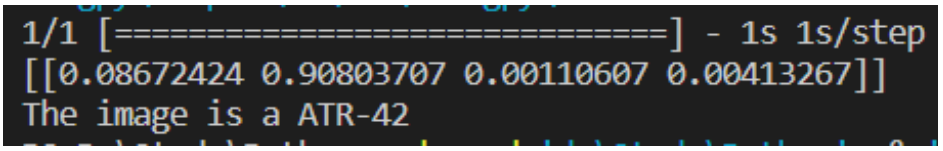


Рисунок 4.28 – Результат класифікації літака ATR-42

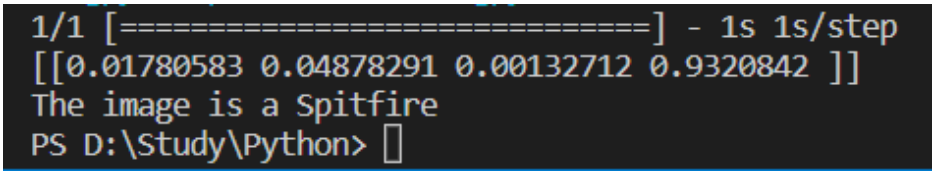


Рисунок 4.29 – Результат класифікації літака Spitfire

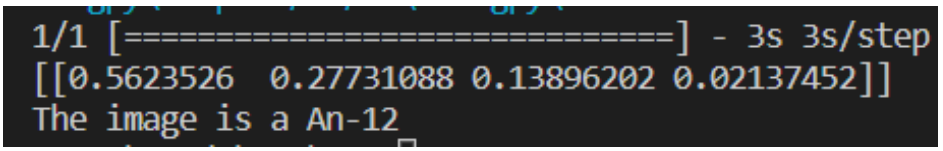


Рисунок 4.30 – Результат класифікації літака Ан-12

Після тестування нейромереж запустимо застосунок та протестуємо його роботу.

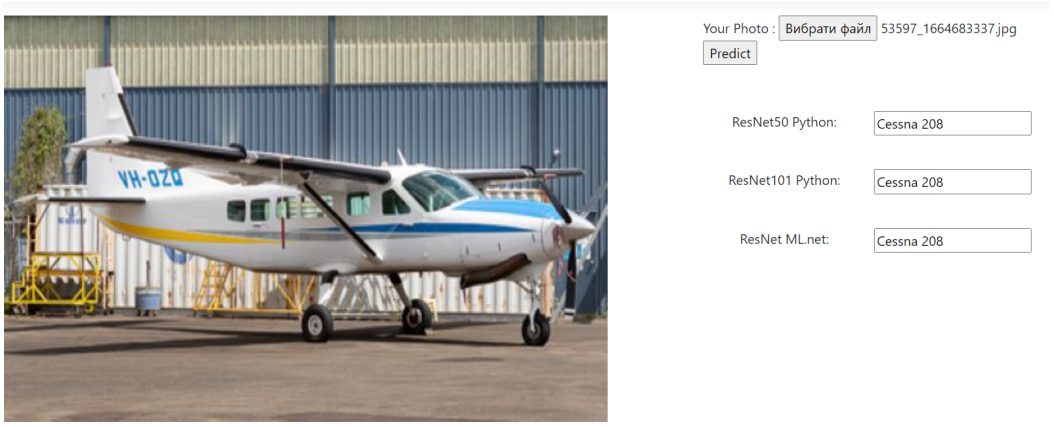


Рисунок 4.31 – Класифікація літака Cessna 208

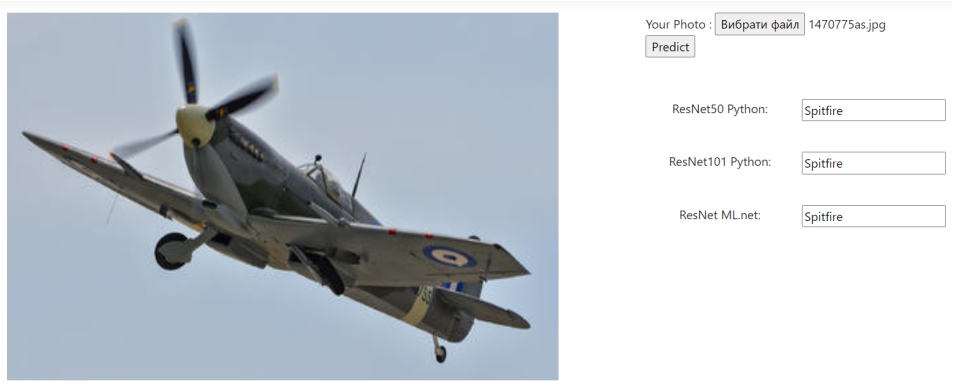


Рисунок 4.32 – Класифікація літака Spitfire

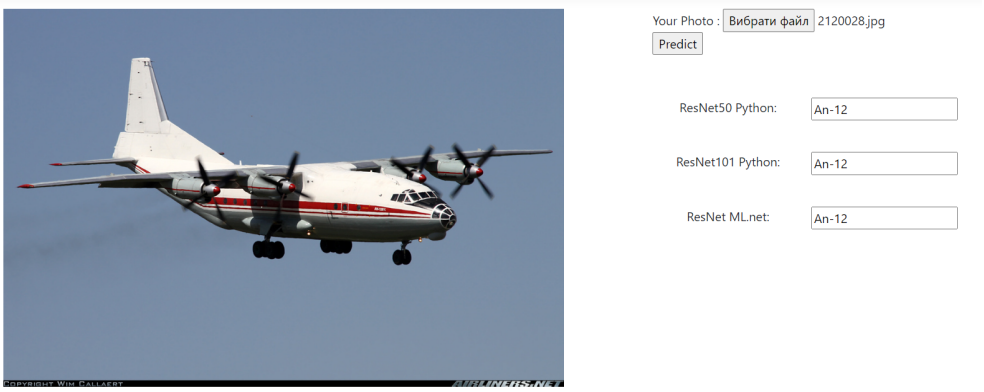


Рисунок 4.33 – Класифікація літака An-12

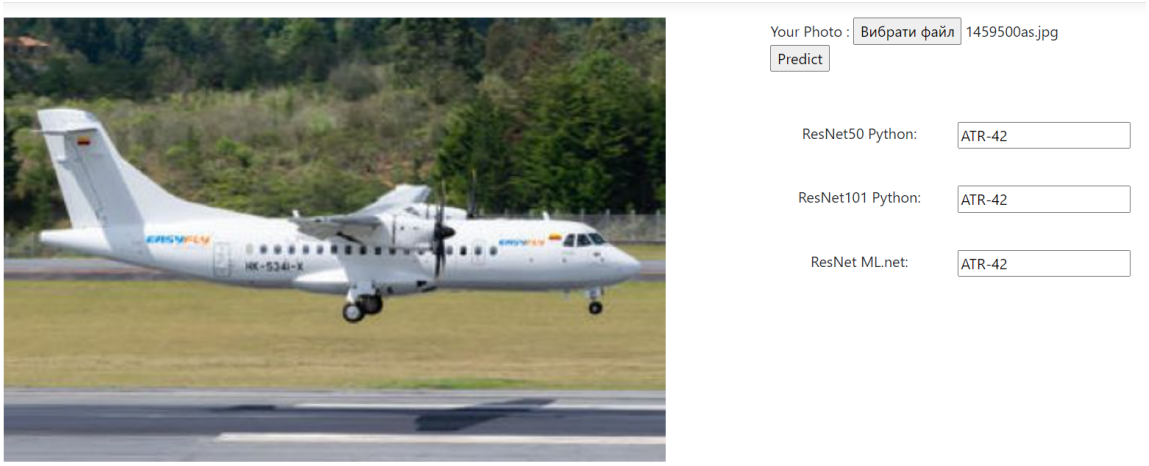


Рисунок 4.34 – Класифікація літака ATR-42

4.6 Аналіз виконаної роботи

Після створення нейромереж та вебзастосунку для перевірки їх роботи, проаналізуємо результат навчання нейромереж. Для цього зробимо діаграму точності навчання нейромереж (рисунок 4.35).

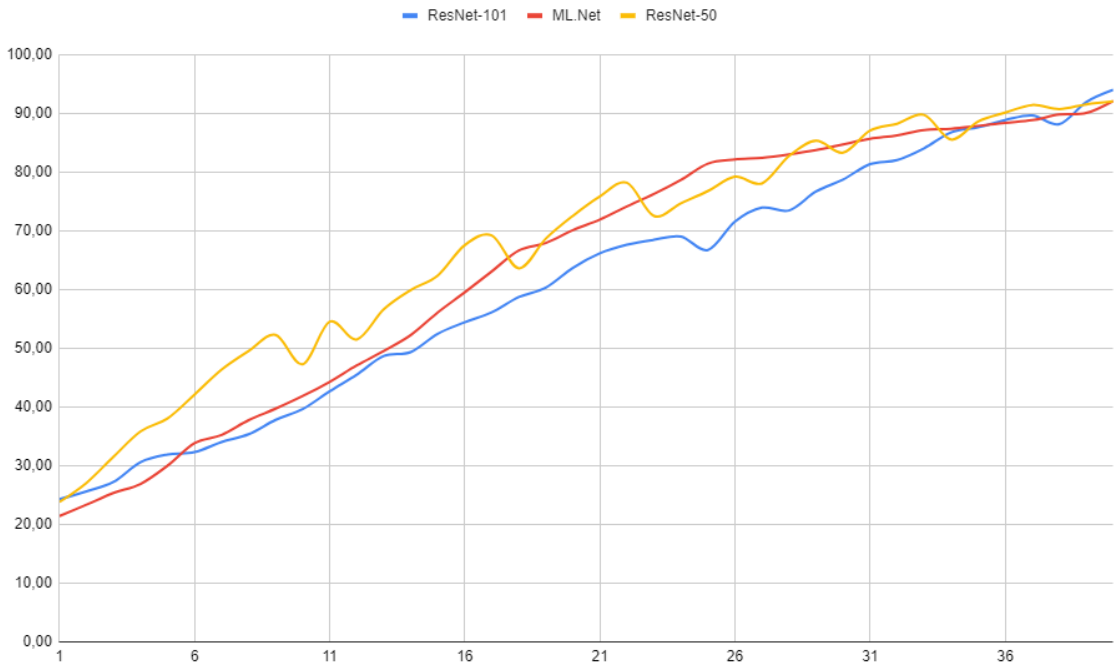


Рисунок 4.35 - Діаграма порівнянь точностей навчання нейромереж на відрізку в 40 епох

На діаграмі можна побачити, що нейромережа ML.Net з архітектурою ResNet-50 найшвидше досягла точності в більш ніж 80%, їй на це знадобилося лише 25 епох, в той час як нейромережа з тією ж архітектурою на Python отримала точність навчання лише на 28 епохі, а нейромережа з архітектурою ResNet-101 на Python - аж на 31 епохі.

Отримані результати навчання нейромережі ResNet-50 (точність 92%) порівняно з іншими подібними дослідженнями [23], в яких для розпізнавання та класифікації використовувалася така ж сама архітектура ResNet-50, у цьому випадку значення точності становили 86%. Це показує, що створена нами нейромережа краще справляється з виконанням даного завдання.

В свою чергу результати навчання нейромережі ResNet-101 (точність 94%) порівняно з іншими подібними дослідженнями [24], в яких для розпізнавання та класифікації використовувалася така ж сама архітектура ResNet-101, у цьому випадку значення точності становили 88,6%. Це показує, що створена нами нейромережа краще справляється з виконанням даного завдання.

Висновки до розділу 4

В даному розділі було розглянуто різні аспекти створення нейромережі ResNet-50 для класифікації фото з використанням різних технологій та інструментів. Було описано два підходи до створення моделі ResNet-50 - за допомогою ML.Net та за допомогою мови програмування Python. Перший підхід дозволяє створити модель в середовищі C# з використанням ML.Net, що забезпечує зручність інтеграції з існуючими додатками. Другий підхід передбачає використання Python для створення моделі з використанням бібліотеки Keras.

Також було описано процес створення зовнішнього сценарію, який дозволяє запускати нейромережі, створені на мові Python, з середовища C#. Це надає можливість використовувати переваги обох мов програмування в одному проекті.

Далі було розглянуто процес створення вебзастосунку, який використовує створену нейромережу для класифікації фото. Веб-застосунок побудований на платформі ASP.NET MVC і надає зручний інтерфейс для користувачів для завантаження та класифікації фотографій.

Нарешті, було проведено тестування нейромережі та вебзастосунку для перевірки їхньої правильності та ефективності. Тестування допомогло переконатися в тому, що створені нейромережі та вебзастосунок працюють.

ВИСНОВКИ

У рамках даної бакалаврської кваліфікаційної роботи було проведено дослідження моделей нейронної мережі ResNet для класифікації об'єктів на фотографіях. В процесі дослідження було створено три моделі ResNet-50 на мові програмування Python, ResNet-101 на мові програмування Python та ResNet-50 на мові програмування C#.

Отримані результати показали, що всі три моделі показали високу точність класифікації об'єктів на фотографіях. Зокрема, найкращий результат було отримано за допомогою моделі ResNet-101, яка показала точність на рівні 94%. Також було проведено порівняння результатів з раніше опублікованими даними, що свідчить про ефективність використання нейронної мережі ResNet для класифікації об'єктів на фотографіях.

Дослідження показало, що найбільш оптимальним варіантом для розробки та тренування нейронних мереж є використання мови програмування Python, оскільки це дозволяє забезпечити високу швидкість розробки та зниження часу тренування мережі. Також було встановлено, що робота нейронної мережі ResNet на мові програмування C# можлива, але потребує додаткових зусиль для її розробки та тренування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

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. Z. Li, K. Jia, Z. Li and M. Yao, "Design and Implementation of Jewelry Selling System Based on .NET MVC Framework," 2012 International Conference on Computer Science and Service System, Nanjing, China, 2012, pp. 1575-1578, doi: 10.1109/CSSS.2012.394.

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. 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.

22. 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.

23. 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.

24. 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.

ДОДАТОК А
Код програмної реалізації

```
public class HomeController : Controller {
        AirCheckModel.ModelInput model = new AirCheckModel.ModelInput();
        public HomeController() {

        }
        public IActionResult Index() {
            return View();
        }
        [HttpPost]
        public JsonResult Prediction(){
            var file = Request.Form.Files[0];
            var path = "C:\\Users\\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[] img = System.IO.File.ReadAllBytes(fullPath);
            string py50 = PredictionPython(fileName, "ResNet50");
            string py101 = PredictionPython(fileName, "ResNet101");
            string mlnet = PredictMLNet(fileName);
            return Json(new{python50=py50,py101=python101,image= img,mlnet=mlnet});
        }
        public string PredictMLNet(string name){
            string imageName = @"C:\Users\BachelorWork\Pictures\" + name;
            System.Drawing.Image image = System.Drawing.Image.FromFile(imageName);
            byte[] imageBytes;
            using (MemoryStream ms = new MemoryStream()){
                image.Save(ms, System.Drawing.Imaging.ImageFormat.Jpeg);
                imageBytes = ms.ToArray();
            }
            model = new AirCheckModel.ModelInput() {
                ImageSource = imageBytes
            };
            var result = AirCheckModel.Predict(model);
            string[] airplanes = new string[] {"An-12","ATR-42","Cessna 208","Spitfire" };
```

```
            int maxInd = result.Score.ToList().IndexOf(result.Score.Max());
            return airplanes[maxInd];
        }
        public string PredictionPython(string name, string type = "ResNet50"){
            var psi = new ProcessStartInfo();
            psi.FileName = "C:\\Python311\\python.exe";
            psi.Arguments = $"D:\\Study\\Python\\NeuralImage\\main.py {type} {name}";
            psi.UseShellExecute = false;
            psi.CreateNoWindow = true;
            psi.RedirectStandardOutput = true;
            psi.RedirectStandardError = true;
            var errors = "";
            var results = "";
            using(var process = Process.Start(psi)){
                errors = process.StandardError.ReadToEnd();
                results = process.StandardOutput.ReadToEnd();
            }
            string res = "";
            try{
                res = results.Split('\n')[3];
            } catch(Exception ex){
                res = "ERROR";
            }
            return res;
        }
        public IActionResult Privacy()
        {
            return View();
        }
        [ResponseCache(Duration = 0, Location=ResponseCacheLocation.None, NoStore=true)]
        public IActionResult Error(){
             return View(new ErrorViewModel{RequestId=Activity.Current.Id});
        }
    }
```