Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Факультет інформаційних технологій
(факультет)

Кафедра системного аналізу та управління
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня бакалавра

Студента Горбенко Максим Михайлович

академічної групи 124-21ск-1

спеціальності 124 Системний аналіз

на тему: «Розробка Telegram бота для розрахунку калорійності страв по фотографії з використанням штучного інтелекту»

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**Інституційною**	
кваліфікаційної роботи	асист. Хабарлак К.С.			
розділів:				
Інформаційно-аналітичний	асист. Хабарлак К.С.			
Спеціальний розділ	асист. Хабарлак К.С.			
Рецензент				
Нормоконтролер	доц. Хом'як Т.В.			

Дніпро
2024

ЗАТВЕРДЖЕНО
завідувач кафедри
Системного аналізу та управління
(повна назва)

_____ *к.т.н., доц. Желдак Т.А.*
(підпис) (прізвище, ініціали)

«____»_____20___року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня *бакалавра*

студенту *Горбенко М.М.* **академічної групи** *124-21ск-1*
спеціальності: *124 Системний аналіз*
на тему: *«Розробка Telegram бота для розрахунку калорійності страв по фотографії з використанням штучного інтелекту»*
затверджену наказом ректора НТУ «Дніпровська політехніка»
від *29.04.2024р. №375-с*

Розділ	Зміст	Терміни виконання
1. Інформаційно-аналітичний розділ	Проаналізувати структуру об'єкта дослідження. Визначити предметну область дослідження та проблему, що розв'язується. Обґрунтувати методи виконання поставлених завдань	11.09.2023-28.01.2024
2. Спеціальний розділ	Розв'язати поставлені задачі: 1) збір набору даних; 2) програмна реалізація навчання нейронної мережі; 3) аналіз якості різних нейронних мереж на цільовій задачі; 4) програмна реалізація Telegram бота для розпізнавання страви по фото з виведенням інформації про калорійність страви.	29.01.2024-05.06.2024

Завдання _____ асист. Хабарлак К.С.
(підпис) (прізвище, ініціали)

Дата видачі: _____04.09.2023_____

Дата подання до екзаменаційної комісії: _____

Прийнято до виконання _____ Горбенко М.М.
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 73с., 31 рис., 7 табл., 2 додатки, 11 джерел

Об'єктом дослідження в роботі є процес розробки програмного додатку до месенджера Telegram, що з використанням нейронної мережі аналізує зображення продукту та визначає їх калорійність, що полегшить слідкування за раціоном.

Предметом дослідження є Telegram–бот, що використовує нейронні мережі для розпізнавання страв на зображеннях та розрахунку їх калорійності.

Метою цієї кваліфікаційної роботи – розробка Telegram-бота, який може аналізувати фотографії страв, розпізнавати їх, визначати їх калорійність і надавати користувачам відповідну інформацію у зручному форматі та полегшити ведення статистики що до харчування.

Методи дослідження: У роботі використано методи машинного навчання, зокрема нейронні мережі для розпізнавання зображень, а також методи обробки зображень та програмування для створення Telegram–бота. Для навчання нейронної мережі використовувалися великі набори даних із зображеннями продуктів харчування та їх характеристиками.

В *інформаційно–аналітичному розділі* наведено аналіз об'єкту дослідження та ключових проблем на ньому. Поставлені задачі дослідження та обрано концепції їх розв'язання.

У *спеціальному розділі* сформовано логіку та алгоритм нейронної мережі для аналізу поставленої задачі, написано програмний додаток для розв'язання існуючої проблеми.

Практична цінність отриманих результатів полягає в тому, що запропонована розроблена система із зручним інтерфейсом скорочує час для пошуку страви та калорійності.

Ключові слова: КОМП'ЮТЕРНИЙ ЗІР, ПРОГРАМА–БОТ, TELEGRAM, НЕЙРОННА МЕРЕЖА, MOBILENETV2.

ABSTRACT

Explanatory Note: 73 pages, 31 figures, 7 tables, 2 appendices, 11 references

The object of this research is the development process of a software application for the Telegram messenger that uses a neural network to analyze product images and determine their caloric content, thereby facilitating diet tracking.

The subject of this research is a Telegram bot that uses neural networks to recognize dishes in images and calculate their caloric content.

The aim of this qualification work is to develop a Telegram bot that can analyze photographs of dishes, recognize them, determine their caloric content, and provide users with relevant information in a convenient format, thereby easing the process of diet statistics tracking.

Research methods: The study employs machine learning methods, particularly neural networks for image recognition, as well as image processing and programming techniques for creating the Telegram bot. Large datasets of food images and their characteristics were used to train the neural network.

The informational-analytical section presents an analysis of the research object and key issues related to it. Research tasks are set, and concepts for their resolution are chosen.

The special section formulates the logic and algorithm of the neural network for solving the stated problem and develops a software application to address the existing issue.

The practical value of the obtained results lies in the fact that the proposed system with a convenient interface reduces the time required to find a dish and its caloric content.

Keywords: COMPUTER VISION, PROGRAM BOT, TELEGRAM, NEURAL NETWORK, MOBILENETV2.

ЗМІСТ

ВСТУП

У наш час дуже зростає популярність здорового способу життя та прагнення людей до контролю за своїм раціоном харчування робить актуальним розробку нових інструментів, які допомагають у цій сфері. Одним із таких інструментів може стати Telegram–бот, який буде використовувати нейрону мережу для розрахунку калорійності страв по фотографії.

Ця кваліфікаційна робота призначена розробці такого Telegram–бота. У даній кваліфікаційній роботі буде розглянуто такі питання як:

– аналіз існуючих методів розрахунку калорійності страв;

– обґрунтування вибору нейронної мережі для розрахунку калорійності страв по фотографії;

– опис архітектури та алгоритму роботи нейронної мережі і Telegram–бота;

– реалізацію Telegram–бота та нейронної мережі на основі вибраної мови програмування.

Актуальність даної теми полягає в тому, що Telegram–бот, який використовує нейронну мережу для розрахунку калорійності страв по фотографії, може стати зручним та доступним інструментом для людей, які хочуть контролювати своє харчування.

Практична цінність роботи полягає в тому, що розроблений Telegram–бот може бути використаний для допомоги людям у підтримці здорового способу життя або контролювати калорійність раціону харчування.

Використання нейронної мережі дозволяє автоматизувати процес розрахунку калорійності страв, що робить його зручним та доступним для широкого кола користувачів.

Очікується, що розроблений Telegram–бот стане корисним інструментом для людей, які прагнуть до здорового способу життя.

Даний проект був опублікований у тезах конференції під назвою «I (VII) міжнародна науково–практична конференція здобувачів вищої освіти і молодих учених «Інформаційні технології: теорія і практика». Тези доповідей (Дніпро 20 – 22 березня 2024).

ІНФОРМАЦІЙНО–АНАЛІТИЧНИЙ РОЗДІЛ

1.1 Постановка задачі

У сучасному світі все більше людей прагнуть вести здоровий спосіб життя, що включає в себе контроль за раціоном харчування. Розрахунок калорійності страв є важливим аспектом цієї задачі, але може бути складним і трудомістким процесом. Telegram–бот, який може автоматично розраховувати калорійність страв по фотографії, може стати цінним інструментом для людей, які хочуть стежити за своїм харчуванням.

Розробити Telegram–бота, який здатний аналізувати фотографії страв, визначити їх складові та розрахувати калорійність страви з використанням методів машинного навчання та комп’ютерного зору.

Спочатку потрібно провести огляд існуючих методів аналізу зображень та розпізнавання предметів на них, також дослідити існуючи програмні засоби для розпізнавання їжі та розрахунку калорій.

Потім слід розробити архітектуру Telegram–бота, а саме створити інтерфейс бота для спілкування з користувачем та спроектувати структури системи для обробки та аналізу зображення.

Створити набір даних для навчання моделі, а саме підготувати набір фотографій страв з відомою калорійністю та навчити модель розпізнавати та класифікувати продукти на фотографії.

Реалізувати алгоритм обробки та аналізу фотографій страв з інтеграцією інструментів комп’ютерного зору для визначення страви на фотографії, а також реалізувати алгоритм для автоматичного розрахунку калорійності страв.

Метою кваліфікаційної роботи є розробка Telegram–бота, який здатний аналізувати зображення страв та автоматично розраховувати їх калорійність, а результати досліджень та розробка будуть використані для покращення здорового способу життя та підтримувати баланс харчування у користувачів.

Також було проаналізовано декілька схожих за принципом Telegram-ботів, які використовують нейронні мережі для розпізнавання страв по зображеннях та виведення кількості калорій. Серед них варто виділити MyFitnessPal Bot [1] та Bitesnap [2].

MyFitnessPal Bot призначений для трекінгу калорійності спожитої їжі та пошуку інформації про продукти з інтеграцією додатку MyFitnessPal. Однією з основних переваг цього Telegram-бота є велика база даних продуктів, яка підтримує різні мови. Це значно спрощує процес введення та пошуку інформації для користувачів з різних країн. Однак, є й недоліки: бот не завжди коректно розпізнає їжу за зображенням, що може призвести до помилок у підрахунку калорій. Крім того, він більше орієнтований на текстове введення даних, ніж на автоматичне розпізнавання за фотографіями.

Bitesnap, інший схожий Telegram-бот, відстежує харчування користувача за допомогою зображень, автоматично підраховує калорії та аналізує поживні речовини. Перевагами Bitesnap є інтуїтивно зрозумілий інтерфейс, який дозволяє легко користуватися ботом навіть новачкам, а також можливість редагування результатів, що дає змогу користувачам коригувати будь-які помилки. Однак, цей бот має й недоліки: обмежений набір продуктів для розпізнавання, що може знизити точність аналізу, та проблеми з точністю розпізнавання, особливо у випадку складних або незвичних страв.

Таким чином, обидва боти мають свої переваги і недоліки. MyFitnessPal Bot виділяється великою базою даних та багатомовною підтримкою, але поступається в точності розпізнавання зображень. Bitesnap, з іншого боку, пропонує зручний інтерфейс та можливість редагування, але обмежена база даних продуктів та точність розпізнавання можуть стати проблемою для деяких користувачів.

Тому було розроблено власний Telegram-бот під назвою PhotoEaterForKkal_Bot, який завдяки використанню сучасних моделей машинного навчання забезпечує більш точне розпізнавання продуктів на

зображеннях порівняно з існуючими аналогами. Наш бот використовує обширну та постійно оновлювану базу даних продуктів та страв, що гарантує точну інформацію про калорійність та поживну цінність їжі.

Однією з головних переваг розробленого Telegram-бота є висока швидкість оброблення зображень, що дозволяє надавати користувачу результати без затримки. Ця оперативність у поєднанні з точністю розпізнавання робить PhotoEaterForKkal_Bot зручним та ефективним інструментом для щоденного використання.

Крім того, бот оснащений інтуїтивно зрозумілим інтерфейсом, який робить процес взаємодії з ним легким та приємним навіть для користувачів, які не мають досвіду використання подібних технологій. База даних бот включає широкий асортимент продуктів та страв з усього світу, що особливо корисно для користувачів з різних культурних середовищ та з різними кулінарними уподобаннями.

Завдяки цим характеристикам, PhotoEaterForKkal_Bot може стати конкурентоспроможним та привабливим для широкого кола користувачів. Висока точність розпізнавання страв, розширена база даних та швидкість обробки даних роблять його надійним помічником у контролі харчування та підтримці здорового способу життя.

1.2 Розробка нейронної мережі

Згорткова нейронна мережа (ConvNet/CNN) – це алгоритм глибокого навчання, який може приймати вхідне зображення, призначати важливість (навчання ваги та зміщення) різним аспектам/об'єктам зображення і мати можливість відрізняти одне від одного. Попередня обробка, необхідна ConvNet, набагато нижче порівняно з іншими алгоритмами класифікації. Хоча у примітивних методах фільтри розробляються вручну, за достатньої підготовки

ConvNets мають можливість вивчити ці фільтри/характеристики. Приклад згорткової нейронної мережі представлено на рисунку 1.1 [3].

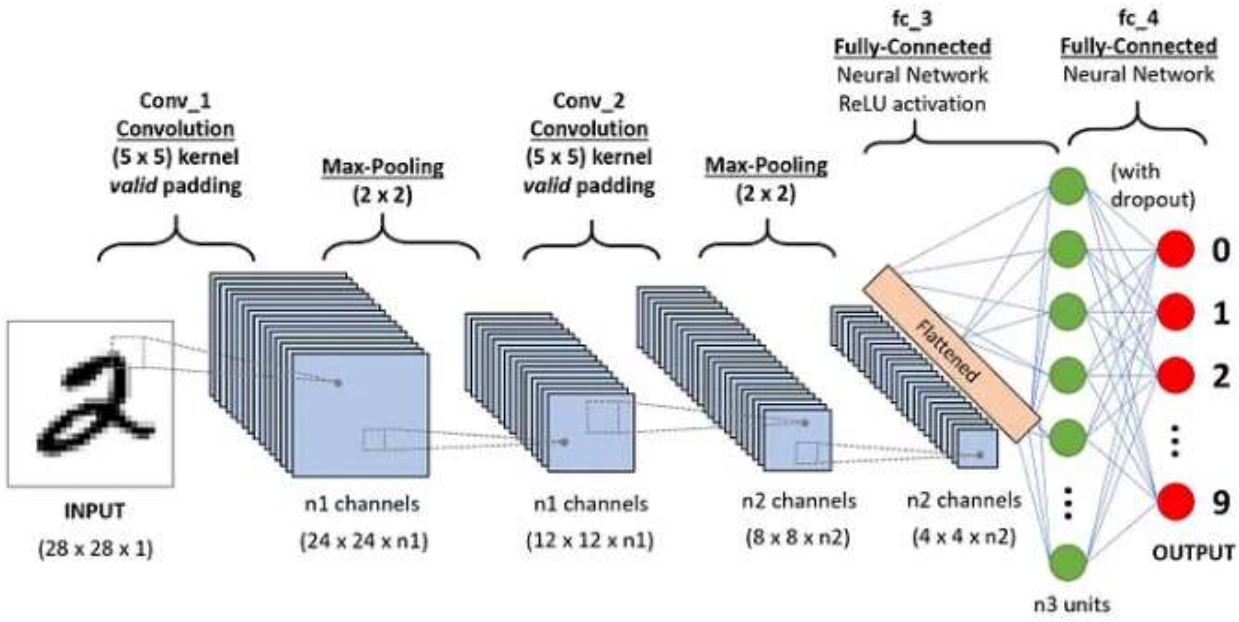


Рисунок 1.1 – Послідовність згорткової нейронної мережі для класифікації рукописних чисел

Згорткова нейронна мережа складається з трьох основних типів блоків:

Перший це – згортковий шар який є основою згорткових нейронних мереж, що робить їх потужним інструментом для аналізу візуальних даних. На відміну від звичайних нейронних мереж, які обробляють дані поточково, згорткові шари використовують операцію згортки, що дозволяє їм ефективно виявляти локальні характеристики вхідних даних.

Як саме працює згортковий шар? В основі роботи згорткового шару лежать різні фільтри, які також називаються ядрами. Кожен фільтр має невеликий розмір, наприклад 3x3 або 5x5 пікселів також фільтри містять вагу, які налаштовуються під час навчання мережі.

Згортка нейронної мережі відбувається коли фільтр ковзає по вхідному зображенню, множачи кожне значення пікселя вхідного зображення на відповідне значення ваги у фільтрі. Потім результати множення додаються разом, створюючи одне значення активації для даної позиції фільтра і цей

процес повторюється для всіх позицій фільтра на зображенні, створюючи карту активацію.

Після створення згортки йде функція активації , яка до карти активації застосовує нелінійну функцію активації, наприклад, ReLU (1.1) або функції sigmoid(1.2) і tanh(1.3), це вводить нелінійність у процес, дозволяючи мережі навчитися більш складним зв'язкам між вхідними даними. Також для функцій активації було створено графік який представлено на рисунку 1.2

$$ReLU(x) = \max(0, x) \tag{1.1}$$

$$sigmoid(x) = \frac{1}{1+e^{-x}} \tag{1.2}$$

$$tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} \tag{1.3}$$

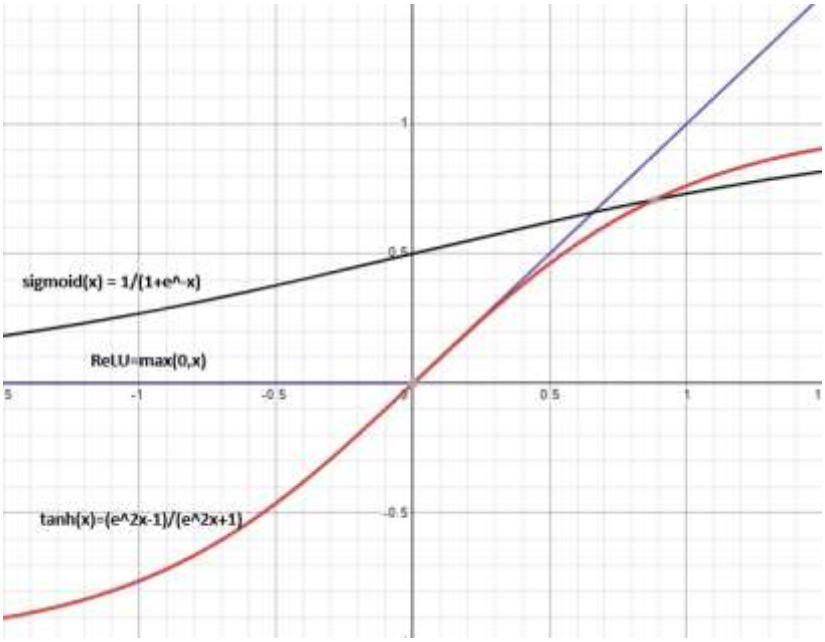


Рисунок 1.2 – Графік функцій активації

Також згортковий шар може мати декілька фільтрів, розташованих один за одним, утворюючи так звану «глибину» при цьому кожен наступний шар використовує карти активації попереднього шару як вхідні дані, дозволяючи мережі вивчити все більш абстрактні характеристики зображень. Приклад

згорткового шару де нейрони згорткового шару (синього), з'єднані з їхнім рецептивним полем (червоним) представлено на рисунку 1.3 [4].

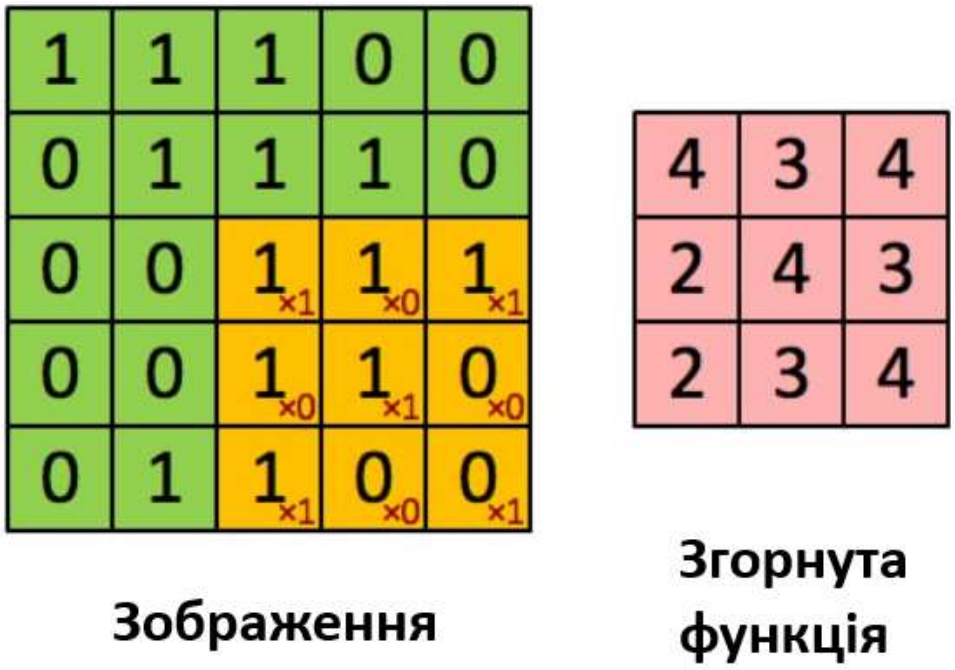


Рисунок 1.3 – Приклад згорткового шару

Після згорткового шару йде шар субдискретизації (пулінг).

Це операція об'єднання яка включає переміщення двовимірного фільтра по кожному каналу карти об'єктів і підсумовування об'єктів, що лежать в області, що охоплюється фільтром.

Для карти об'єктів, що має розміри n_h xn_w xnc розміри вихідних даних, отриманих після шару об'єднання, рівні (1.4)

$$\frac{(n_{h-f+1})}{s} * \frac{(n_w-f+1)}{s} * n_c \qquad (1.4)$$

, де n_h – висота карти об'єктів; n_w – ширина картки об'єктів; n_c – кількість каналів у карті об'єктів; f – розмір фільтра; s – довжина кроку.

Загальна архітектура моделі CNN полягає в тому, щоб мати кілька шарів згортки та об'єднання, розташовані один за одним.

Навіщо використовувати шари пулінгу?

Шари об'єднання використовуються для зменшення розмірів карток об'єктів. Таким чином, це зменшує кількість параметрів, які необхідно вивчити, та обсяг обчислень, що виконуються в мережі.

Шар об'єднання підсумовує об'єкти, які є в області карти об'єктів, створеної шаром згортки. Таким чином, подальші операції виконуються над об'єктами, а не над точно позиціонованими об'єктами, створеними шаром згортки. Це робить модель стійкішою до змін становища об'єктів на вхідному зображенні.

Також шар субдискретизації (пулінг) має декілька типів, а саме:

Субдискретизація із функцією максимуму, що являє собою операцію об'єднання, яка вибирає максимальний елемент з області карти об'єктів, що охоплюється фільтром. Таким чином, вихідні дані після шару максимального пулу будуть картою об'єктів, що містить найбільш помітні функції попередньої карти об'єктів. Приклад представлено на рисунку 1.4

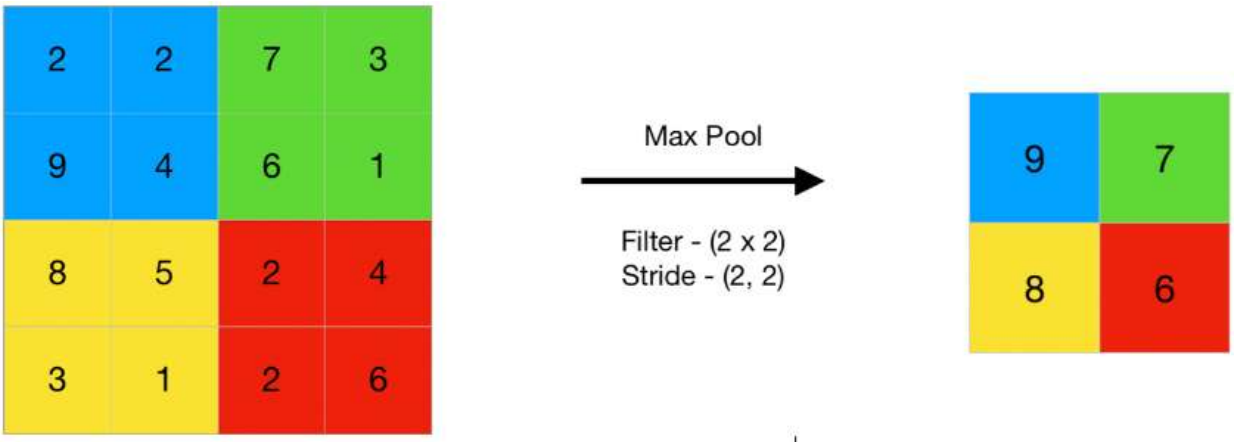


Рисунок 1.4 – Шар субдискретизації із функцією максимуму

Другим типом об’єднання є субдискретизації із функцією середнього що, обчислює середнє значення елементів, що є в області карти об'єктів, охопленої фільтром. Отже, тоді як максимальне об'єднання дає найбільш помітний об'єкт у конкретній ділянці карти об'єктів, середнє об'єднання дає середнє значення функцій, присутніх у патчі. Приклад представлено на рисунку 1.5.

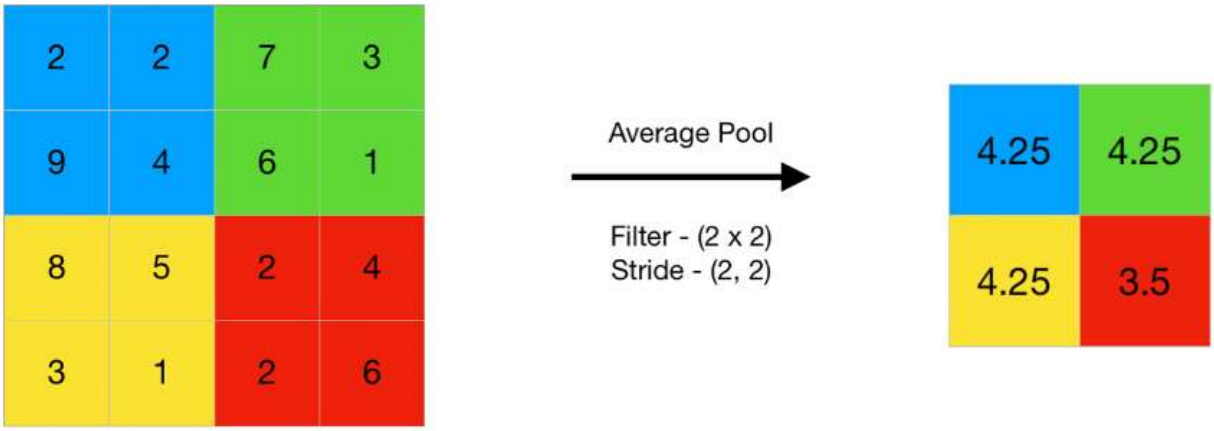


Рисунок 1.5 – Шар субдискретизації із функцією середнього

У згорткових нейронних мережах (CNN) шар об'єднання – це поширений тип шару, який зазвичай додається після згорткових шарів. Шар об'єднання використовується для зменшення просторових розмірів (тобто ширини та висоти) карток об'єктів при збереженні глибини (тобто кількості каналів).

Шар об'єднання працює шляхом поділу вхідної картки об'єктів на набір регіонів, що не перетинаються, званих регіонами об'єднання. Кожен регіон об'єднання потім перетворюється на одне вихідне значення, яке відображає наявність певного об'єкта у цьому регіоні. Найбільш поширеними типами операцій об'єднання є максимальне об'єднання та середнє об'єднання.

При максимальному об'єднанні вихідне значення для кожної області об'єднання є просто максимальним значенням вхідних значень у цьому регіоні. Це призводить до збереження найважливіших особливостей у кожній області об'єднання та відкидання менш важливої інформації. Максимальне об'єднання часто використовується в CNN для розпізнавання об'єктів, оскільки воно допомагає ідентифікувати найбільш відмінні особливості об'єкта, такі як його краї і кути.

При середньому об'єднанні вихідне значення кожної області об'єднання є середнє значення вхідних значень у цьому регіоні. Це призводить до збереження більшої кількості інформації, ніж при максимальному об'єднанні, але також може послабити найважливіші функції. Середнє об'єднання часто

використовується в CNN для таких завдань, як сегментація зображень і виявлення об'єктів, де потрібно детальніше подавати вхідні дані.

Шари об'єднання зазвичай використовуються разом з згортковими шарами CNN, при цьому кожен шар об'єднання зменшує просторові розміри карт об'єктів, в той час як згорткові шари витягують з вхідних даних все більш складні функції. Отримані карти об'єктів потім передаються повністю зв'язний шар, який виконує остаточну задачу класифікації або регресії [5].

Після створення згорткового шару та шару пулінгу потрібно створити повнозв'язковий шар.

Нейронні мережі є набір залежних нелінійних функцій. Кожна окрема функція складається з нейрону (або перцептрону). У пов'язаних шарах нейрон застосовує лінійне перетворення до вхідного вектора за допомогою матриці ваг. Потім продукт застосовується нелінійне перетворення за допомогою нелінійної функції активації f (1.5) .

$$y_{jk} = f(\sum_{i=1}^{n_H} w_{jk} x_i + w_{j0}) \quad (1.5)$$

, де y_{ik} – вихід нейрона k на шарі i, w_{ik} – вага, яка з'єднує вхід x_i нейроном k на шарі i, x_i – значення вхідного сигналу i, w_{i0} – вага зсуву.

Тут ми беремо скалярний добуток матриці ваг W та вхідного вектора x. Член зміщення (W0) можна додати до нелінійної функції. Я проігнорую це до кінця статті, оскільки це не впливає на розміри виведення чи ухвалення рішень і є просто ще однією вагою.

Якщо ми візьмемо шар пов'язаної нейронної мережі з вхідним розміром дев'ять і вихідним розміром чотири, операцію можна візуалізувати наступним чином представлено на рисунку 1.6.

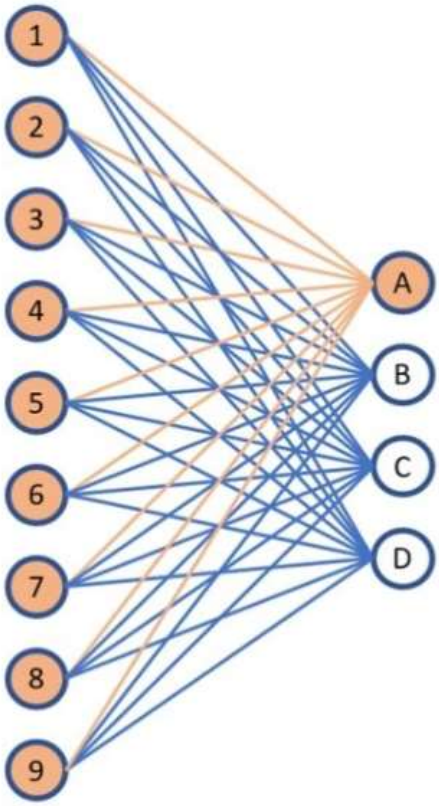


Рисунок 1.6 – Приклад повнозв’язного слою

На зображенні (рис. 1.6) вище показано, чому називаються такі шари «повністю пов'язаними» або іноді «щільно пов'язаними». Є всі можливі зв'язки між шарами, що означає, що кожен вхід вхідного вектора впливає на кожен вихід вихідного вектора. Однак не всі ваги впливають на всі вихідні дані. На зображенні є лінії між кожним вузлом вище. Помаранчеві лінії представляють перший нейрон (або перцептрон) шару. Ваги цього нейрона впливають тільки вихід A і впливають виходи B, C чи D [6].

1.3 Технологія нейронної мережі

При створенні нейронної мережі необхідно визначитись з моделлю, яка буде використовуватися для її навчання. У цьому випадку розглядалися дві моделі: MobileNetV2 та ResNet.

MobileNetV2 – це легка архітектура згорткової нейронної мережі (CNN) спеціально розроблена для мобільних і вбудованих програм машинного зору. Дослідники Google розробили її як удосконалення вихідної моделі MobileNet

[7]. Ще одним чудовим аспектом цієї моделі є її здатність забезпечувати гарний баланс між розміром моделі та точністю, що робить її ідеальною для пристроїв з обмеженими ресурсами. Приклад архітектури нейронної мережі MobileNetV2 представлено на рисунку 1.7.

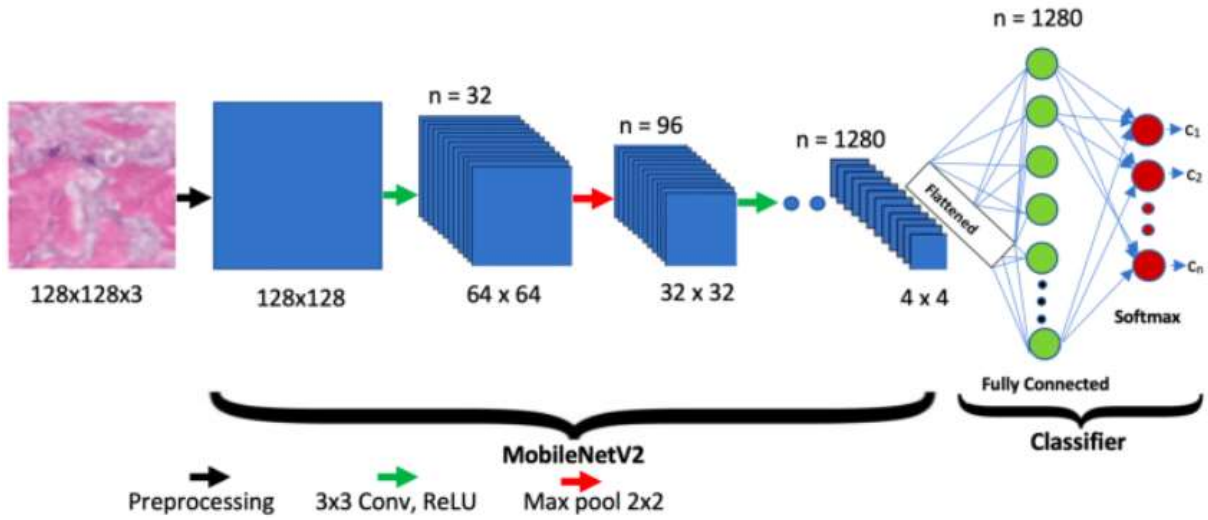


Рисунок 1.7 – Архітектура нейронної мережі MobileNetV2

Ключовою особливістю MobileNetV2 є включення кількох ключових функцій, які підвищують його ефективність і результативність у задачах класифікації зображень. Ці функції включають відокремлену по глибині згортку, інвертовані залишки, проектування вузьких місць, лінійні вузькі місця та блоки стиснення та збудження (SE). Кожна з цих функцій відіграє вирішальну роль у зниженні обчислювальної складності моделі за збереження високої точності.

Навіщо використовувати MobileNetV2 для класифікації зображень?

Використання MobileNetV2 для класифікації зображень дає кілька переваг. По–перше, його легка архітектура дозволяє ефективно розгортати його на мобільних та вбудованих пристроях з обмеженими обчислювальними ресурсами. По–друге, MobileNetV2 забезпечує конкурентоспроможну точність порівняно з більшими та більш дорогими в обчислювальному відношенні

моделями. Нарешті, невеликий розмір моделі дозволяє скоротити час виведення, що робить її придатною для додатків реального часу.

Архітектура MobileNetV2 складається з серії згорткових шарів, за якими слідують відокремлені по глибині згортки, інвертовані залишки, проектування вузьких місць, лінійні вузькі місця та блоки стиснення та збудження (SE). Ці компоненти працюють разом, щоб зменшити кількість необхідних параметрів і обчислень, зберігаючи здатність моделі фіксувати складні функції.

Перший шар архітектури MobileNetV2, є глибоко відокремлена згортка – це метод, який використовується в MobileNetV2 для зниження обчислювальних витрат на згортки. Він розділяє стандартну згортку на дві окремі операції: глибинну згортку та точкову згортку. Такий поділ значно скорочує кількість необхідних обчислень, роблячи модель ефективнішою.

Другий шар є інвертованими залишками який є ключовим компонентом MobileNetV2, який допомагає підвищити точність моделі. Вони вводять структуру «вузьких місць», яка збільшує кількість каналів перед застосуванням відокремлених по глибині згорток. Це розширення дозволяє моделі охопити складніші функції та підвищити ефективність її подання.

Третім шаром є проектуванням вузького місця MobileNetV2 яке додатково знижує обчислювальні витрати за рахунок використання згорток 1×1 для зменшення кількості каналів перед застосуванням згорток, що розділяються по глибині. Такий вибір конструкції допомагає підтримувати гарний баланс між розміром моделі та точністю.

Перевагами MobileNetV2 є:

– Має значно меншу кількість параметрів, що робить її ідеальною для мобільних та вбудованих пристроїв;

– Завдяки своїй легкій структурі MobileNetV2 може виконувати розрахунки значно швидше, ніж інші моделі;

– Незважаючи на свою легкість, MobileNetV2 демонструє високу точність на задачах комп'ютерного зору;

– MobileNetV2 можна налаштувати для різних задач комп'ютерного зору за допомогою зміни її розміру та архітектури.

MobileNetV2 – потужна та легка модель для класифікації зображень. Його ефективна архітектура у поєднанні зі здатністю підтримувати високу точність робить його ідеальним вибором для пристроїв з обмеженими ресурсами. Розуміючи ключові функції, архітектуру, процес навчання, оцінку продуктивності та реалізацію MobileNetV2, розробники та дослідники можуть використовувати його можливості для ефективного вирішення реальних завдань класифікації зображень [8].

ResNet (скорочення від Residual Network) – це тип архітектури нейронної мережі який був розроблений для вирішення проблеми зникнення градієнтів у глибоких нейронних мережах, які заважали їхній роботі при вирішенні великомасштабних завдань розпізнавання зображень.

У цьому посібнику детально обговорюється архітектура ResNet, включаючи її історію, ключові функції та програми у різних галузях.

Архітектура ResNet зазвичай ділиться чотирма частини, кожна з яких містить кілька залишкових блоків різної глибини. Перша частина мережі складається з одного згорткового шару, за яким слідує максимальне об'єднання для зменшення просторових розмірів вхідних даних. Друга частина Мережі містить 64 фільтри, а третя та четверта частини – 128 і 256 фільтрів відповідно. Остання частина мережі складається з глобального середнього пулу і пов'язаного рівня, який виробляє вихідні дані.

Залишкове навчання – це концепція, яка була введена в архітектуру ResNet для вирішення проблеми зникнення градієнта. У традиційних глибоких нейронних мереж кожен шар застосовує набір перетворень до вхідних даних для отримання вихідних даних. ResNet вводить залишкові з'єднання, які дозволяють мережі вивчати залишкові зіставлення, які являють собою різницю між вхідними та вихідними даними шару.

Залишкові сполуки формуються шляхом додавання вхідних даних до вихідних рівнів, що дозволяє градієнтам текти безпосередньо через мережу без послаблення. Це дозволяє Мережі вивчати залишкове зіставлення, використовуючи коротке з'єднання, що минає перетворення шару.

Архітектура ResNet складається з кількох рівнів, кожен із яких містить залишкові блоки. Залишковий блок – це набір шарів, які виконують набір перетворень на вході для отримання вихідних даних і включають з'єднання швидкого доступу, яке додає вхідні дані до вихідних.

Архітектура ResNet–50 є одним із найпопулярніших варіантів і складається з п'яти етапів, кожен з яких містить кілька залишкових блоків. Перший етап складається з згорткового шару, за яким слідує шар максимального пулу, який зменшує просторові розміри вхідних даних.

Другий етап містить три блоки, кожен з яких містить два згорткових шари і з'єднання швидкого доступу. Третій, четвертий і п'ятий етапи містять чотири, шість і три залишкові блоки відповідно. Кожен блок цих етапах містить кілька згорткових шарів і ярлик з'єднання.

Вихідні дані останнього етапу передаються глобальний середній рівень пула, який зменшує просторові розміри карт об'єктів до одного значення для кожного каналу. Вихідні дані шару глобального середнього пулу потім передаються повнозв'язний рівень з активацією softmax, що створює остаточний результат роботи мережі.

ResNet має кілька переваг, які роблять його популярним вибором для програм глибокого навчання:

– ResNet дозволяє створювати більш глибокі нейронні мережі з більш як сотнею шарів, що раніше було неможливо через проблему зникнення градієнта. Залишкові зв'язки дозволяють мережі краще вивчати уявлення та оптимізувати градієнтний потік, що спрощує навчання глибших мереж;

– ResNet досягла найвищого рівня продуктивності на кількох еталонних наборах даних, таких як ImageNet, CIFAR–10 і CIFAR–100, продемонструвавши свою точність порівняно з іншими архітектурами глибоких нейронних мереж;

– ResNet забезпечує більш швидку збіжність під час навчання завдяки залишковим з'єднанням, які забезпечують кращий градієнтний потік та оптимізацію. Це призводить до більш швидкого навчання та кращої збіжності до оптимального рішення;

– ResNet підходить для трансферного навчання, дозволяючи повторно використовувати раніше використану мережу.

Незважаючи на численні переваги, ResNet має кілька недоліків, які слід враховувати:

– ResNet – це складна архітектура, що вимагає більше пам'яті та обчислювальних ресурсів, ніж дрібніші мережі. Це може бути обмеженням у сценаріях з обмеженими ресурсами, такими як мобільні пристрої або вбудовані системи;

– ResNet може бути схильний до переоснащення, особливо якщо мережа надто глибока або набір даних невеликий. Це можна пом'якшити за допомогою методів регуляризації, таких як відсів або за рахунок використання мереж меншого розміру з меншою кількістю шарів;

– Інтерпретованість ResNet може бути складною, оскільки Мережа вивчає складні та абстрактні уявлення, які важко зрозуміти. Це може бути обмеженням у сценаріях, де інтерпретованість має вирішальне значення, наприклад, при медичній діагностиці або виявленні шахрайства.

ResNet – це потужна архітектура глибоких нейронних мереж, яка зробила революцію в галузі комп'ютерного зору, дозволивши створювати більш глибокі та точні мережі. Його залишкові зв'язки забезпечують кращий градієнтний потік та оптимізацію, що спрощує навчання більш глибоких мереж та підвищує продуктивність на еталонних наборах даних.

Однак у ResNet є обмеження, такі як складність, схильність до перенавчання і обмежена інтерпретованість. При виборі ResNet або будь–якої іншої архітектури глибоких нейронних мереж для конкретного завдання слід враховувати ці недоліки [9].

MobileNetV3 – це архітектура нейронної мережі, розроблена для ефективного застосування в мобільних і вбудованих пристроях, де обчислювальні ресурси обмежені. MobileNetV3 є наступником MobileNetV1 та MobileNetV2 і включає покращення для підвищення продуктивності та ефективності [10].

В цій кваліфікаційній роботі використовується нейрона мережа MobileNetV2. Її вибір обумовлений кількома факторами а саме:

– по–перше простота навчання. MobileNetV2 значно легше налаштувати та запустити ніж ResNet;

– по–друге швидкість навчання. MobileNetV2 навчається значно швидше, ніж ResNet, що економить час та ресурси;

– по–третє мобільність. MobileNetV2 значно менш вимоглива до обчислювальних потужностей, що робить її ідеальною для використання на мобільних пристроях а ResNet, навпаки, потребує потужного комп’ютеру для роботи.

Отже для кваліфікаційної роботи було використано саме MobileNetV2 так як він є оптимальним вибором для цієї роботи завдяки його простоті, швидкості та мобільності.

СПЕЦІАЛЬНИЙ РОЗДІЛ

2.1 Розробка бази даних

Для навчання нейронної мережі необхідно зібрати великий обсяг даних про страви, на яких вона буде навчатися. Зазвичай це включає в себе фотографії страв з різних джерел. Для збору даних про страви було використано фотографії з інтернету, таких як онлайн кулінарні рецепти, а також було розроблено спеціального бота для Telegram, який збирав дані від користувачів. Такий бот запитував користувачів надсилати фотографії страв із певними хештегами, щоб збирати якісні дані для навчання. Це допомагає забезпечити різноманітність та представництво в навчальному наборі даних, що важливо для ефективного навчання нейронної мережі [11].

Було зібрано понад 3300 різних фотографій страв для навчання нейронної мережі. З них близько 2250 фотографій було отримано з інтернету, а решта – близько 1050 зібрано завдяки користувачам через Telegram-бота.

Процес знаходження даних про страви з інтернету представлено на рисунку 2.1

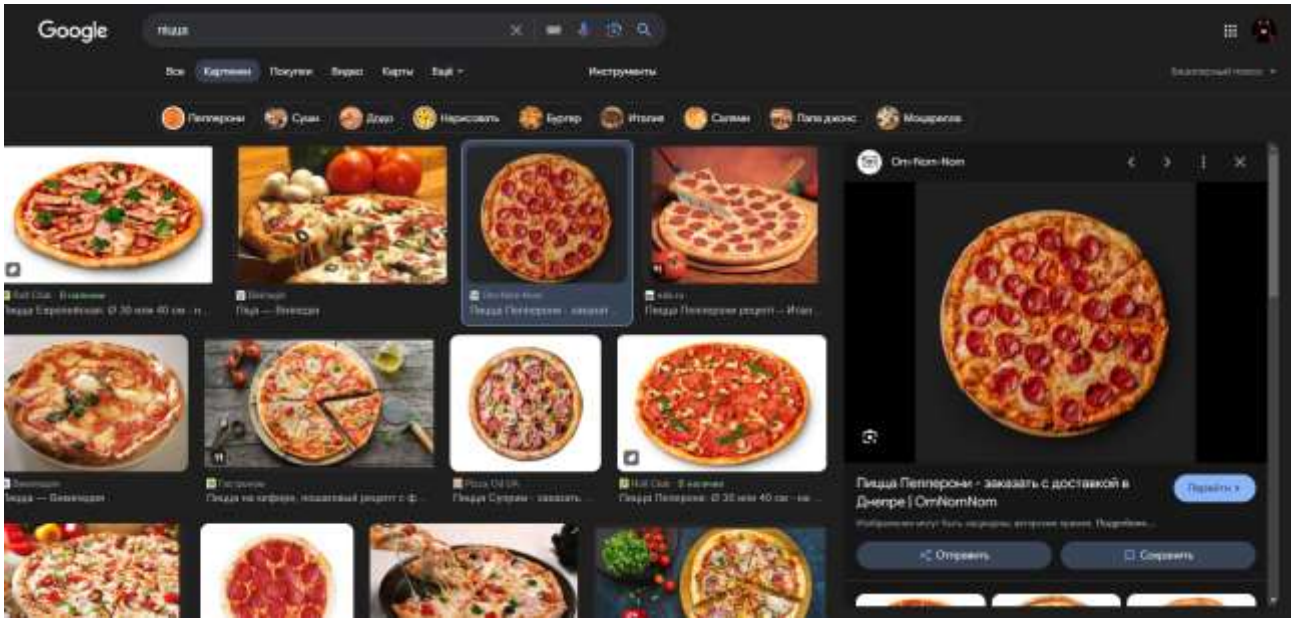
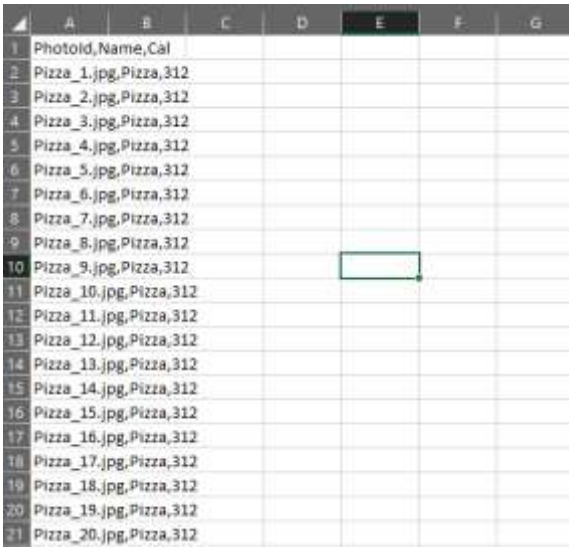


Рисунок 2.1 – Знаходження даних про страви

Процес збору даних від користувачів через Telegram-бота, який був реалізований за допомогою мови програмування Python та бібліотеки TeleBot для створення ботів. Приклад збору даних від користувачів за допомогою Telegram-бота представлено на рисунку 2.2

Рисунок 2.2 – Збір даних від користувачів за допомогою Telegram–бота

Після збору фотографій було створено файл, який містив в собі інформацію про кожну фотографію. Ця інформація включала назву фотографії, яка слугувала ідентифікатором, назву страви, зображеної на фотографії, а також кількість калорій на 100 грамів цієї страви. Цей файл із зібраною інформацією дозволяв ефективно управляти та аналізувати дані під час навчання нейронної мережі. Файл з інформацією про страви представлено на рисунку 2.3



	A	B	C	D	E	F	G
1	PhotoId,Name,Cal						
2	Pizza_1.jpg,Pizza,312						
3	Pizza_2.jpg,Pizza,312						
4	Pizza_3.jpg,Pizza,312						
5	Pizza_4.jpg,Pizza,312						
6	Pizza_5.jpg,Pizza,312						
7	Pizza_6.jpg,Pizza,312						
8	Pizza_7.jpg,Pizza,312						
9	Pizza_8.jpg,Pizza,312						
10	Pizza_9.jpg,Pizza,312						
11	Pizza_10.jpg,Pizza,312						
12	Pizza_11.jpg,Pizza,312						
13	Pizza_12.jpg,Pizza,312						
14	Pizza_13.jpg,Pizza,312						
15	Pizza_14.jpg,Pizza,312						
16	Pizza_15.jpg,Pizza,312						
17	Pizza_16.jpg,Pizza,312						
18	Pizza_17.jpg,Pizza,312						
19	Pizza_18.jpg,Pizza,312						
20	Pizza_19.jpg,Pizza,312						
21	Pizza_20.jpg,Pizza,312						

Рисунок 2.3 – Файл з інформацією про страви

Після збору всіх даних страви було необхідно розподілити на окремі класи. Як результат, було створено 121 клас страв, кожен з яких відображає певний тип страви і для кожного класу зібрано понад 30 різних фотографій страв. Цей розподіл було документовано в окремому файлі, зображеному на рисунку 2.4.

Рисунок 2.4 – Файл з класами

За допомогою класифікації за типом страв, представленою у таблиці 2.1, було створено діаграму, яка ілюструє кількість страв кожного типу. Ця діаграма, зображена на рисунку 2.5, надає візуальне уявлення про розподіл страв за їхнім призначенням.

Таблиця 2.1

Класифікація страв по типам

Тип страви	**Кількість страв**	**Загальна кількість зображень**
Перші страви	10	320
Другі страви	23	710
Гарніри	7	240
Закуски	14	380
Десерти	17	530
Напої	12	360
Інше	22	700
Загалом	**121**	**3300**

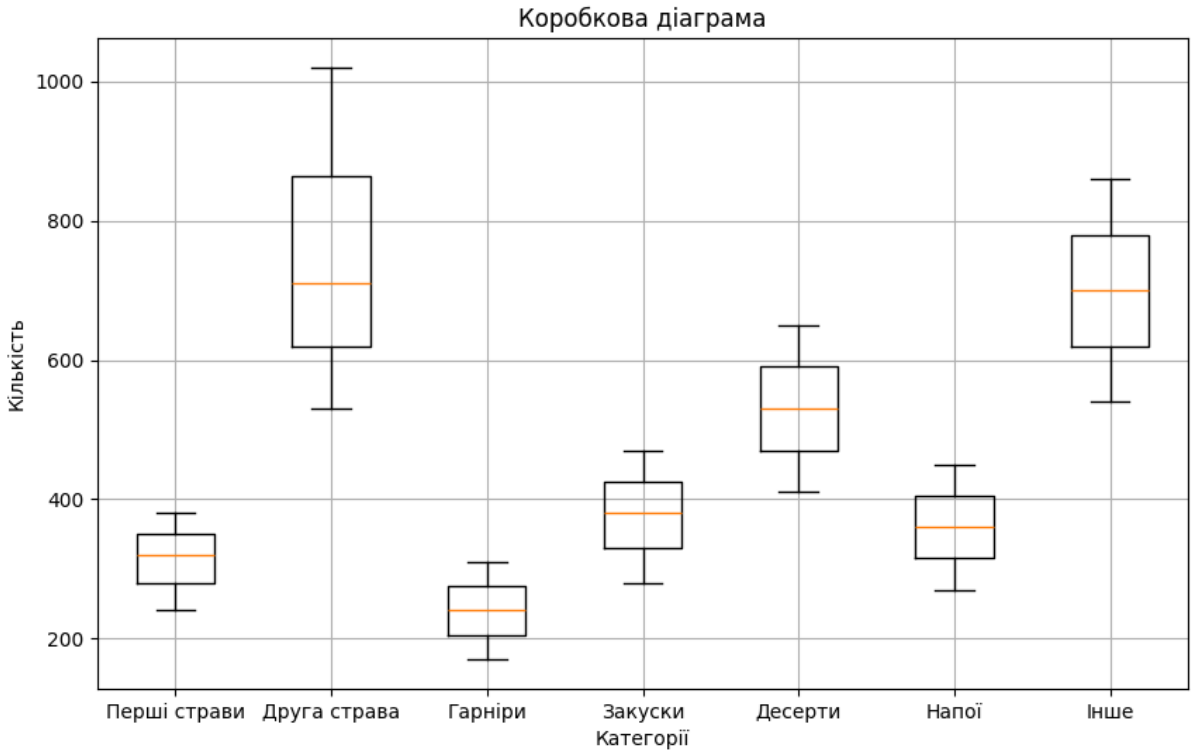


Рисунок 2.5 – Кількість зображень по кожному типу страв

Після того, як дані було оброблено та класифіковано, наступним кроком є розділення їх на навчальну та тестову вибірку для подальшої перевірки ефективності моделі. Для цього використовується стратегія, де 80% від усіх даних відводиться на навчальну вибірку, щоб модель могла вивчати шаблони та залежності в даних. Решта 20% відводиться на тестову вибірку, щоб перевірити, наскільки добре модель може узагальнювати нові дані, які вона раніше не бачила. Таким чином, дані розбиваються на дві вибірки: 2626 припадає на навчальну вибірку, а 657 - на тестову. Цей підхід дозволяє ефективно оцінити точність та надійність моделі перед її застосуванням на нових даних.

Вибірки було створено шляхом випадкового розподілу даних за допомогою функції, наведеної у лістингу 2.1.

Лістинг 2.1 – Функція для розподілу даних між вибірками

```
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)
```

2.2 Аналіз навчання нейронної мережі

Першим етапом у процесі навчання нейронної мережі є вибір оптимальної моделі. У випадку з нашим вибором, ми мали чотири варіанти: MobileNet, MobileNetV2, MobileNetV3Small та MobileNetV3Large. Кожна з цих моделей має свої унікальні особливості, але для нашої конкретної задачі ми хочемо вибрати ту, яка забезпечить найвищу точність під час навчання.

Одним з ключових критеріїв для вибору є рівень точності моделі під час навчання. Ми хочемо, щоб наша модель могла правильно класифікувати дані з якомога вищою точністю. Тому, перш ніж перейти до наступних етапів, таких як підготовка даних та навчання моделі, ми повинні обрати ту нейронну мережу, яка демонструє найкращі результати.

Отже, обираючи між MobileNet, MobileNetV2, MobileNetV3Small та MobileNetV3Large, ми маємо врахувати цілі нашого проекту, обмеження ресурсів та потреби щодо точності моделі.

Розрахунки проведено з використанням повнозв'язного шару, що мав 256 нейронів та кроком навчання в 0.001 протягом 15 епох. Результати розрахунків наведені в таблиці 2.2, а також представлено на рисунку 2.6 у вигляді діаграми, яка відображає рівень точності для кожної моделі.

Таблиця 2.2

Результати рівня точності для кожної моделі

Тип моделі	Точність навчання	Точність валідації
MobileNet	60,06%	62,10%
MobileNetV2	63,21%	68,49%
MobileNetV3Small	41,43%	59,97%
MobileNetV3Large	60,40%	72,60%

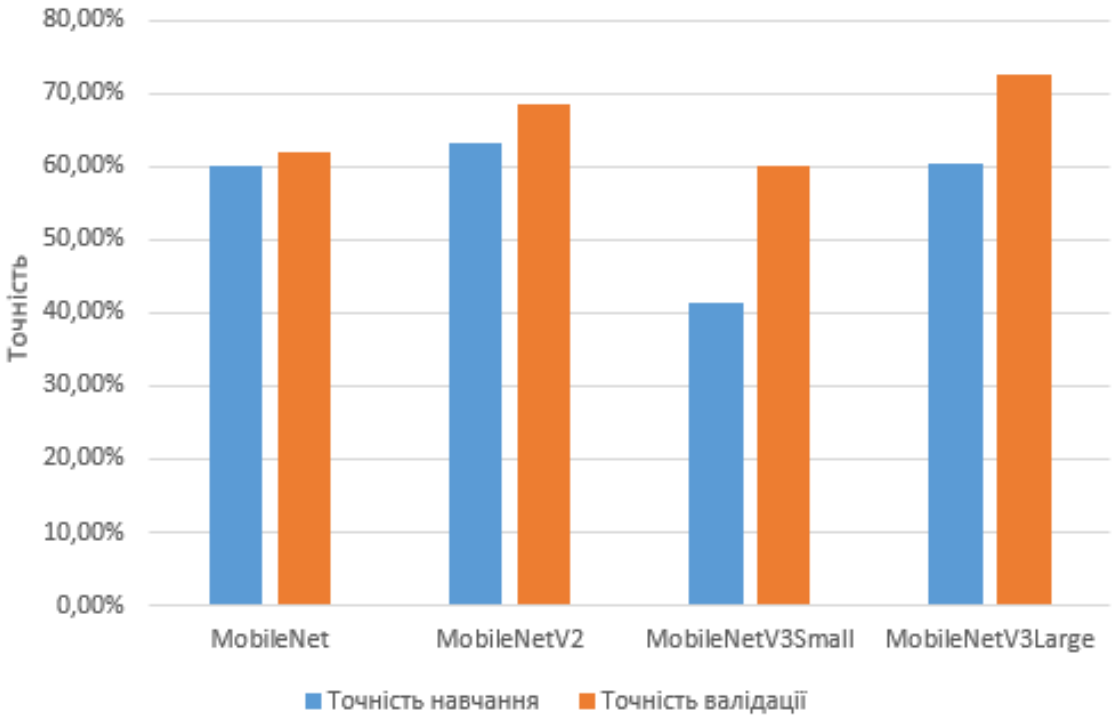


Рисунок 2.6 – Діаграма навчання нейронної мережі для різних моделей

Після аналізу було вирішено обрати модель MobileNetV2, оскільки її точність навчання одна із кращих. Проте також можливо розглянути модель MobileNetV3Large, оскільки її точність навчання була вищою за MobileNetV2. Це вказує на те, що обидві моделі можуть бути ефективними в даному контексті, але MobileNetV3Large продемонструвала трохи кращі результати, що може бути важливим фактором при роботі проте MobileNetV2 аналізує дані швидше ніж MobileNetV3Large тому подальші аналізи проводилися для MobileNetV2 але також було проведено аналіз і для MobileNetV3Large.

Другим критичним етапом у процесі навчання нейронної мережі є вибір оптимальної кількості нейронів у повнозв'язаному шарі, оскільки це безпосередньо впливає на кінцеву точність моделі. Аналіз проводився з різними значеннями кількості нейронів: 32, 64, 128, 256 та 512 з використання моделі MobileNetV2 протягом 15 епох навчання. Результати навчання представлені у таблиці 2.3, а також графічно відображені на рисунку 2.7.

Таблиця 2.3

Аналіз кількості нейронів у повнозв'язаному шарі для моделі MobileNetV2

Кількість нейронів в повнозв’язному шарі	Точність навчання	Точність валідації
32 нейрона	22,43%	53,42%
64 нейрона	44,38%	60,73%
128 нейрона	57,12%	68,34%
256 нейрона	63,21%	68,49%
512 нейрона	67,85%	68,32%

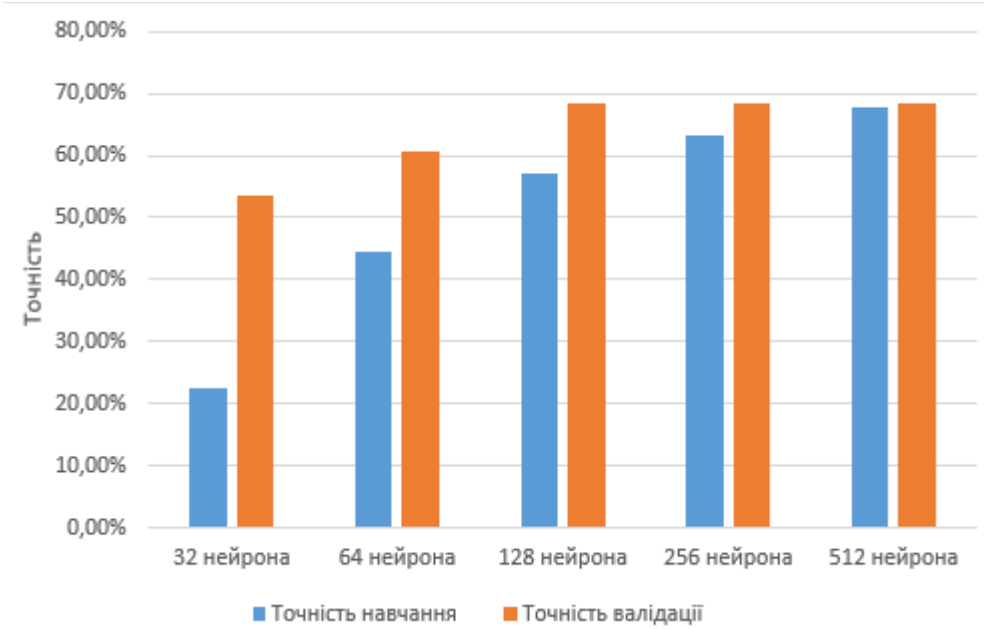


Рисунок 2.7 – Діаграма аналізу кількості нейронів на навчання нейронної мережі MobileNetV2

Після проведення аналізу було встановлено, що кількість нейронів в мережі суттєво впливає на її кінцеву точність під час навчання. Зокрема, порівняння різних варіантів показало, що модель з 256 нейронами досягає найвищої точності. Однак, варіанти з 128 і 512 нейронами також розглядалися, адже вони відносно близькі до 256 за точністю, але мають меншу втрату під час навчання. Хоча модель з 128 нейронами має меншу втрату, обрано 256 нейронів через їхній більший потенціал впливу на кінцевий результат.

Було проведено додатковий аналіз для моделі MobileNetV3Large з використанням тих самих параметрів навчання, що і для попередньої моделі, з метою визначення її меж продуктивності. Результати цього навчання представлені в таблиці 2.4. Крім того, ці результати графічно відображені на рисунку 2.8.

Таблиця 2.4

Аналіз кількості нейронів у повнозв'язаному шарі для моделі MobileNetV3Large

Кількість нейронів в повнозв’язному шарі	Точність навчання	Точність валідації
32 нейрона	29,06%	57,99%
64 нейрона	47,56%	67,73%
128 нейрона	60,40%	72,60%
256 нейрона	67,75%	74,28%
512 нейрона	74,41%	75,65%

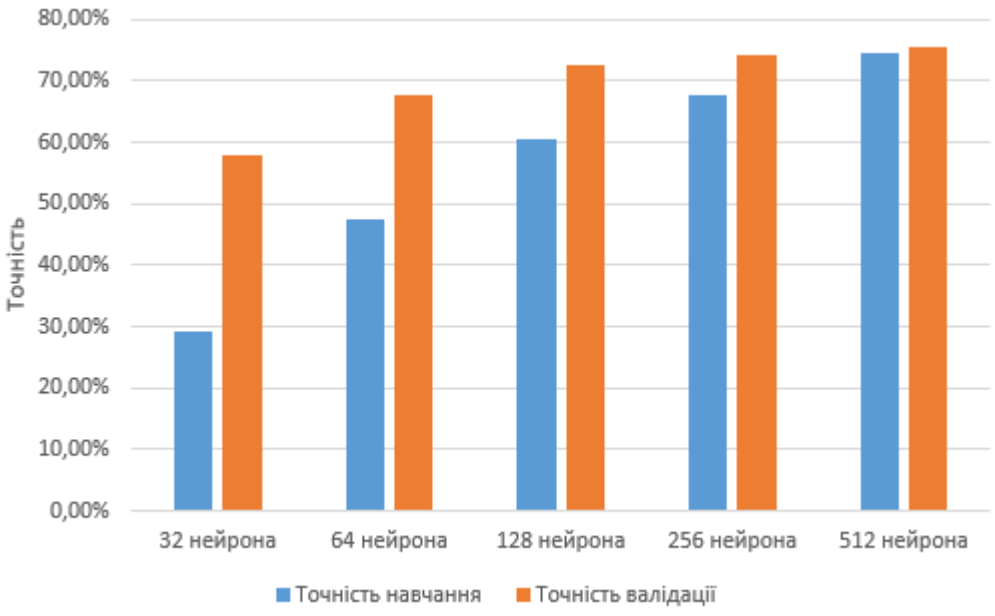


Рисунок 2.8 – Діаграма аналізу кількості нейронів на навчання нейронної мережі для MobileNetV3Large

Для моделі MobileNetV3Large найбільш оптимальним виявилось використання 516 нейронів на повнозв’язному шарі, оскільки це дало найвищу точність навчання серед інших конфігурацій. Однак, варто зазначити, що конфігурація з 256 нейронами також продемонструвала достатньо високу точність, що робить її прийнятною альтернативою. Використання меншої кількості нейронів, наприклад, 256, може бути доцільним у випадках, коли є обмеження щодо обчислювальних ресурсів або потрібно зменшити час навчання моделі. Це свідчить про гнучкість MobileNetV3Large і дозволяє налаштовувати її параметри в залежності від конкретних вимог до задачі та наявних ресурсів.

На третьому етапі навчання нейронної мережі важливо вибрати оптимальний крок навчання. Цей параметр визначає, наскільки швидко або повільно мережа адаптується до навчальних даних. Для аналізу ефективності різних значень кроку навчання були вибрані такі значення: 1.0, 0.5, 0.1, 0.01, 0.001, 0.0005, 0.0001, використовуючи модель MobileNetV2 та кількість 256 нейронів на повнозв’язному шару протягом 15 епох навчання.

Мета цього аналізу – знайти оптимальний крок навчання, який дозволяє швидко досягти високої точності навчання моделі. Результати цього аналізу представлені в таблиці 2.5 і графічно на рисунку 2.9.

Таблиця 2.5

Аналіз кроку навчання MobileNetV2

Крок навчання	Точність навчання	Точність валідації
Крок 1.0	1,21%	1,06%
Крок 0.5	1,32%	0,60%
Крок 0.1	1,49%	0,90%
Крок 0.01	8,57%	20,40%
Крок 0.001	53,59%	67,28%
Крок 0.0005	53,81%	65,60%

Крок 0.0001	35,00%	48,40%

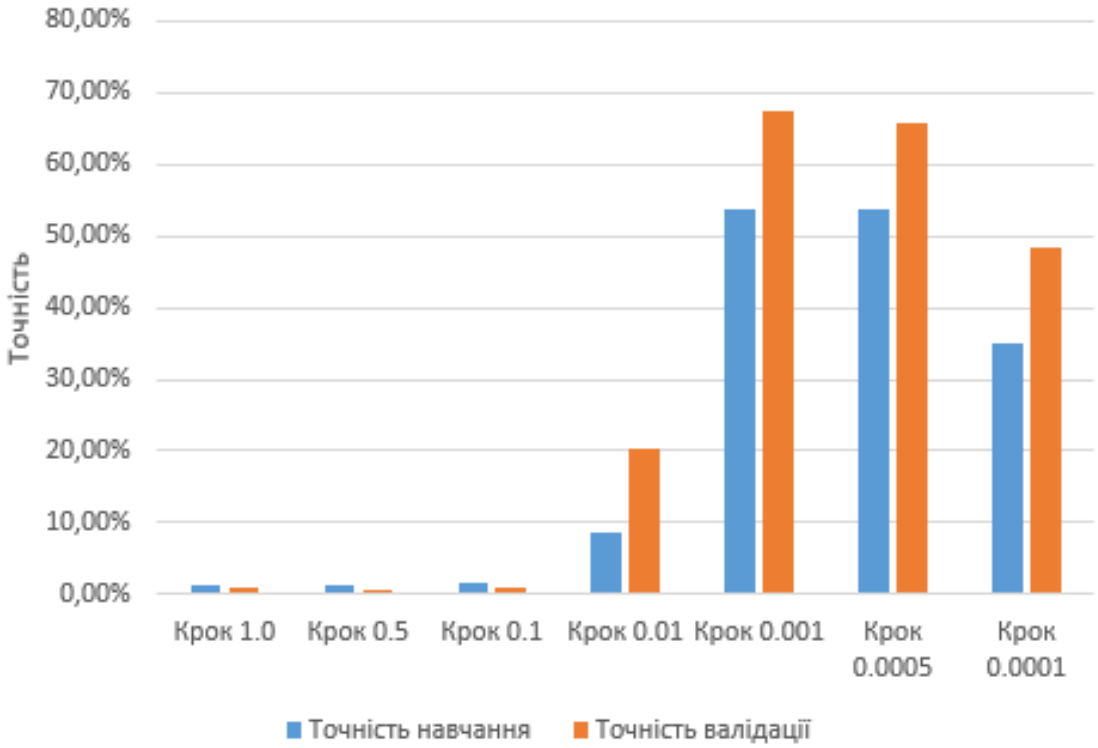


Рисунок 2.9 – Діаграма аналізу кроку навчання на навчання нейронної мережі MobileNetV2

Після аналізу було вирішено застосувати крок навчання 0.001, оскільки саме при його використанні отримано найвищий відсоток точності навчання. Проте, варто відзначити, що також можливо розглянути використання кроку навчання 0.0005, оскільки він наближений за відсотком точності навчання.

Було також проведено аналіз кількості кроків для моделі MobileNetV3Large, щоб визначити, який крок навчання є найбільш ефективним для цієї моделі. Для забезпечення точності та порівнянності результатів, в аналізі використовувалися ті самі параметри, що і для моделі MobileNetV2. Результати цього аналізу детально представлені у таблиці 2.6, де можна побачити залежність продуктивності моделі від кількості кроків навчання. Графічне відображення цих результатів наведено на рисунку 2.10, що дозволяє краще зрозуміти динаміку змін точності та втрат при різних кроках навчання.

Таблиця 2.6

Аналіз кроку навчання MobileNetV3Large

Крок навчання	Точність навчання	Точність валідації
Крок 1.0	0,99%	0,46%
Крок 0.5	0,69%	1,07%
Крок 0.1	1,41%	1,37%
Крок 0.01	25,17%	46,58%
Крок 0.001	60,40%	72,60%
Крок 0.0005	52,21%	69,86%
Крок 0.0001	12,45%	19,03%

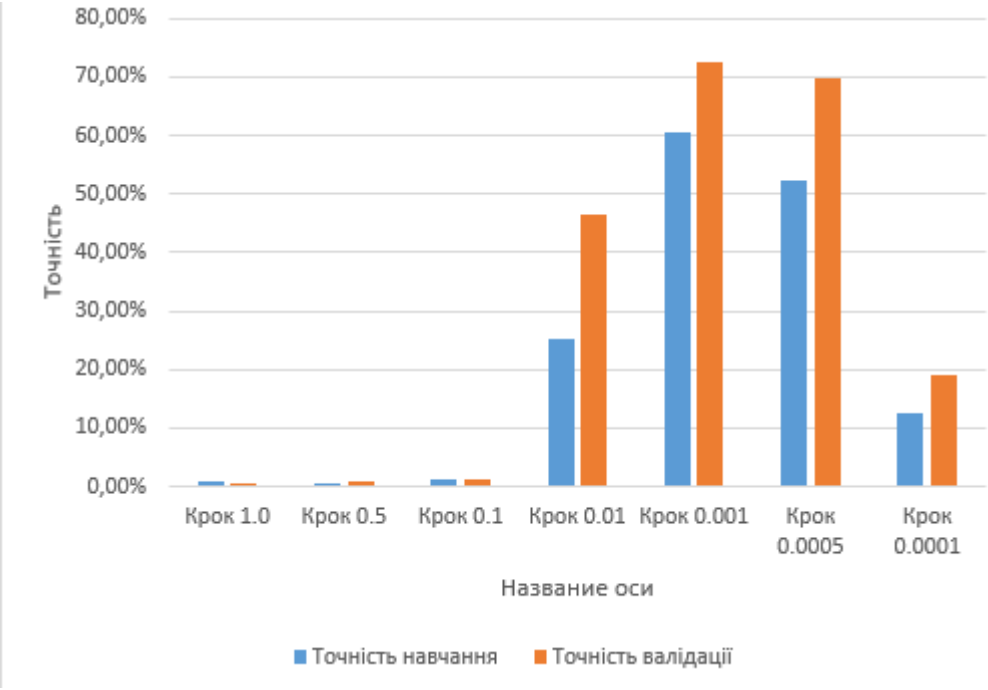


Рисунок 2.9 – Діаграма аналізу кроку навчання на навчання нейронної мережі MobileNetV3Large

Після проведеного аналізу було виявлено, що крок навчання 0.001 забезпечив найвищий відсоток точності для моделі MobileNetV3Large.

Водночас, крок 0.0005 також показав високий рівень точності, наближаючись до результатів, отриманих з кроком 0.001. Це свідчить про те, що обидва параметри кроку навчання є ефективними для цієї моделі. Застосування кроку 0.001 дозволяє досягти високої точності за меншу кількість епох, що може бути корисним для прискорення процесу навчання. Проте, використання кроку 0.0005 може бути доцільним у випадках, коли потрібна додаткова стабільність і поступове вдосконалення моделі, оскільки менший крок може зменшити ризик перенавчання і сприяти кращій узагальненій здатності моделі.

Після ретельного аналізу було визначено, що найоптимальнішими параметрами для навчання нейронної мережі є використання моделей MobileNetV2 або MobileNetV3Large. Для MobileNetV2 було встановлено, що оптимальним налаштуванням є повнозв’язний шар на 256 нейронів, тоді як для MobileNetV3Large – 512 нейронів. Крім того, крок навчання 0.001 був визнаний найефективнішим, оскільки він забезпечив найвищу точність навчання, що є критично важливим для правильної роботи нейронної мережі.

Однак, було обрано саме MobileNetV2, оскільки вона демонструє більшу швидкість навчання порівняно з MobileNetV3Large. Це особливо важливо при використанні нейронної мережі на смартфонах, де обчислювальні ресурси обмежені. Завдяки меншій витраті ресурсів, MobileNetV2 є більш підходящим варіантом для мобільних пристроїв, що робить її кращою у цьому конкретному випадку.

2.3 Аналіз роботи нейронної мережі

Після завершення навчання нейронної мережі необхідно провести її тестування на різних стравах, щоб оцінити її здатність розпізнавати ці страви з певним відсотком точності. Цей етап є критично важливим для визначення ефективності моделі в реальних умовах. Тестування включає перевірку

нейронної мережі на наборі зображень різних страв, не використаних у процесі навчання, щоб оцінити її узагальнюючу здатність та точність класифікації.

Процес перевірки дозволяє визначити, з яким відсотком точності модель може ідентифікувати ті чи інші страви, що дає змогу оцінити її практичну корисність. Результати тестування допоможуть виявити сильні та слабкі сторони моделі, а також дозволять внести необхідні корективи в її архітектуру або параметри навчання для подальшого покращення точності.

Аналіз було проведено на декількох даних один із прикладів представлено на рисунку 2.10.

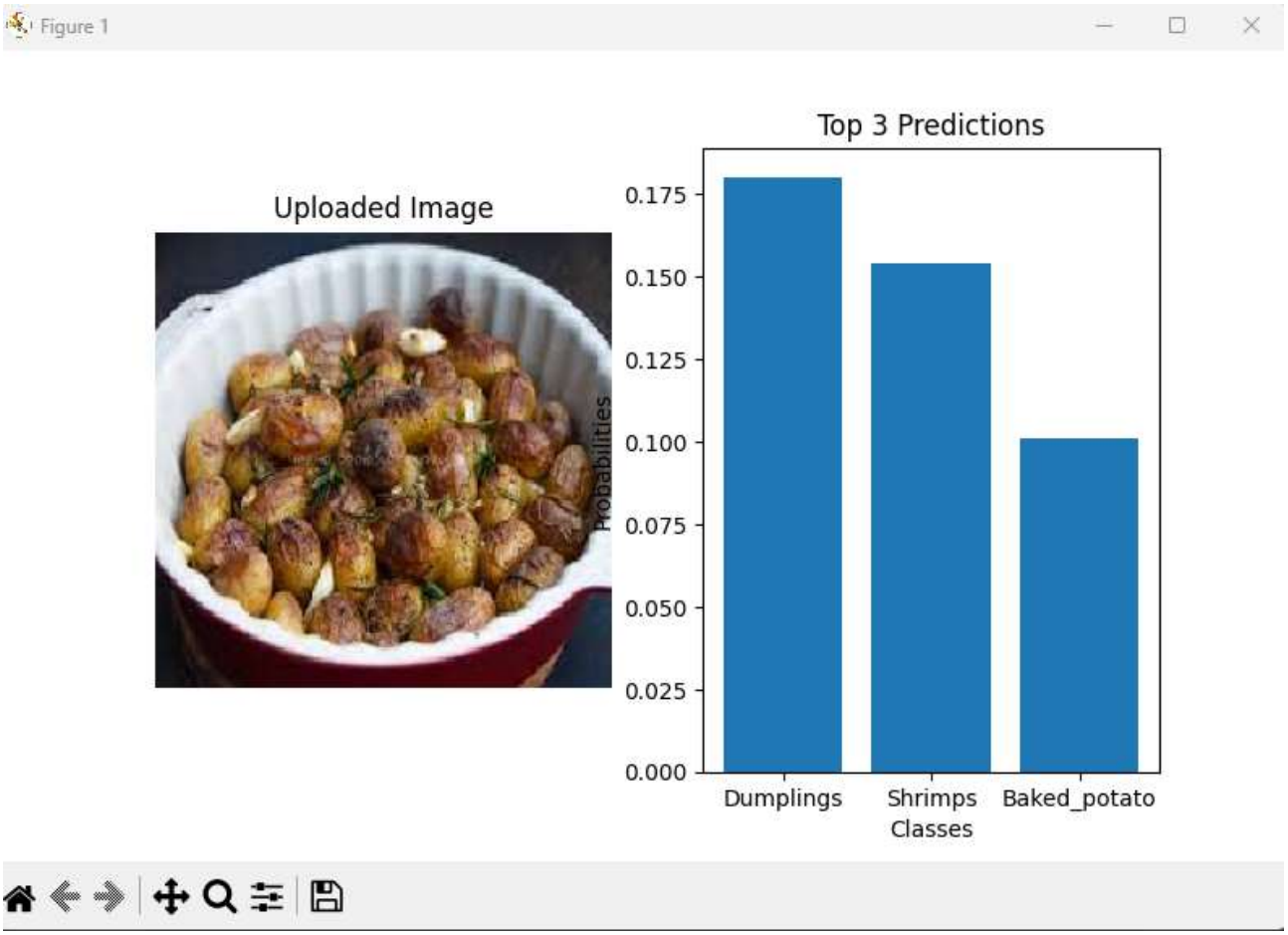


Рисунок 2.10 – Аналіз страви та виведення результату

На зображенні була страва «Запечена картопля», але через погану якість фотографії нейронна мережа неправильно класифікувала її. Нейронна мережа з 17.5% впевненістю визначила, що це «Пельмені», а також з 15% впевненістю

припустила, що на зображенні можуть бути «Креветки». Хоча правильна класифікація «Запечена картопля» також була присутня серед варіантів, але лише з 10% впевненістю. Це сталося через те, що якість фотографії була настільки низькою, що мережа надала перевагу іншим, більш ймовірним, на її думку, варіантам, таким як «Пельмені» та «Креветки», замість справжнього зображення запеченої картоплі.

Також однією з ситуацій, де нейронна мережа помилилася через схожість страв, можна побачити на рисунку 2.11.

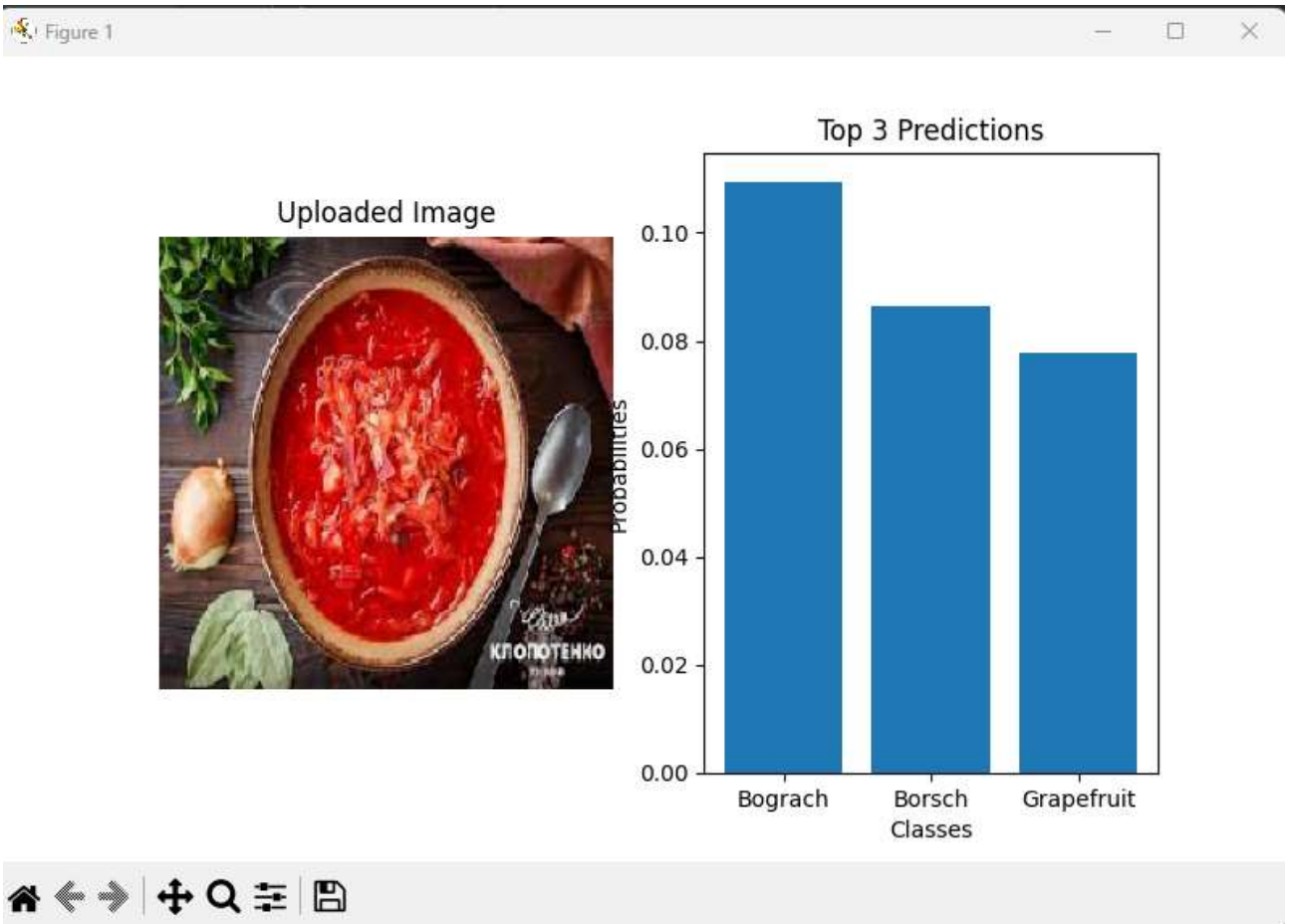


Рисунок 2.11 – Аналіз страви та помилка нейронної мережі

У даному випадку нейронна мережа розпізнала на фото страву під назвою «Бограш», хоча насправді на фото був зображений «Борщ». Ця помилка сталася через велику схожість цих двох страв, що ускладнює точне визначення, яка саме страва зображена на фотографії.

Бограш і борщ мають схожі інгредієнти та зовнішній вигляд, що призводить до труднощів у класифікації навіть для передових нейронних мереж. В обох стравах використовуються схожі види м'яса та овочів, вони мають подібний колір та консистенцію.

Також нейронна мережа можу дуже чітко розпізнавати страви і приклад даної страви представлено на рисунку 2.12 – 2.14

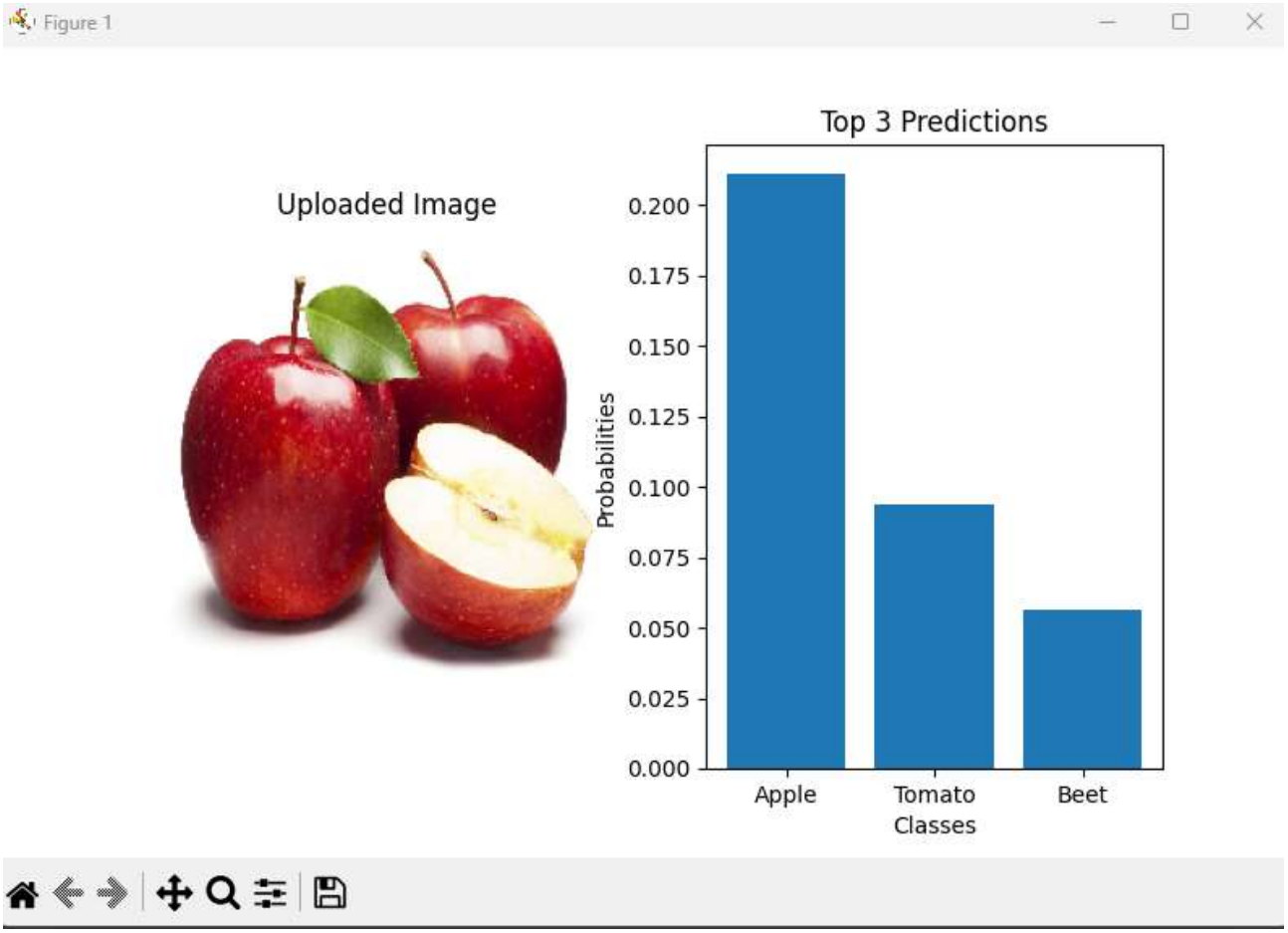


Рисунок 2.12 – Аналіз страви та виведення правильного результату

Також нейронна мережа може розпізнавати об'єкти на фото з дуже високою точністю. Наприклад, у випадку, наведеному вище, мережа правильно ідентифікувала «Яблуко» на зображенні з 20% впевненістю, що вдвічі більше, ніж для наступного варіанту в списку.

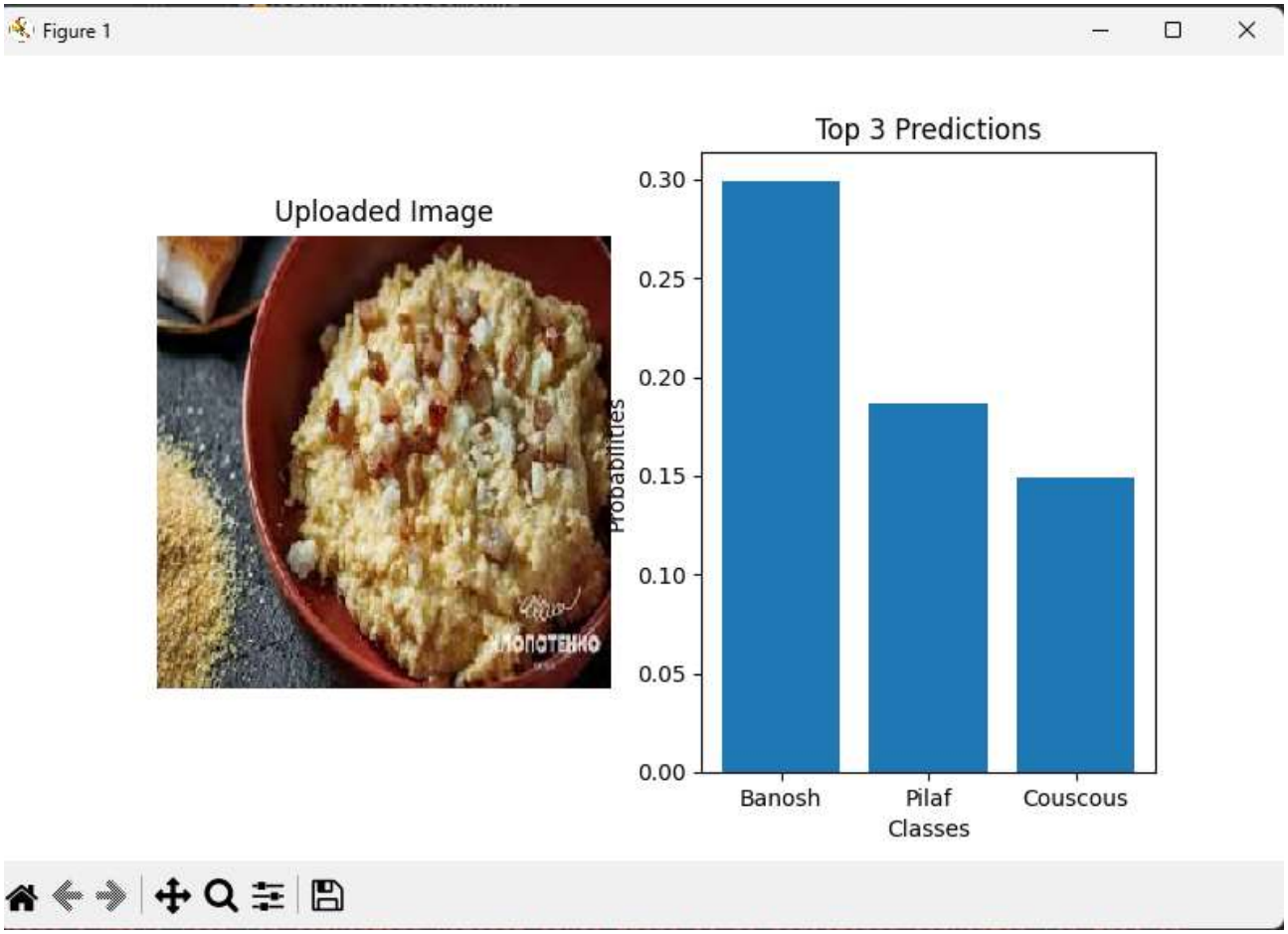


Рисунок 2.13 – Аналіз страви та виведення правильного результату

Одним із прикладів високої точності розпізнавання об’єктів нейронною мережею є ситуація, коли мережа правильно ідентифікувала «Банош» на зображенні з 30% впевненістю. Це майже вдвічі більше, ніж для наступного варіанту в списку, хоча на фотографії можна було б подумати, що зображена «Пшенична каша».

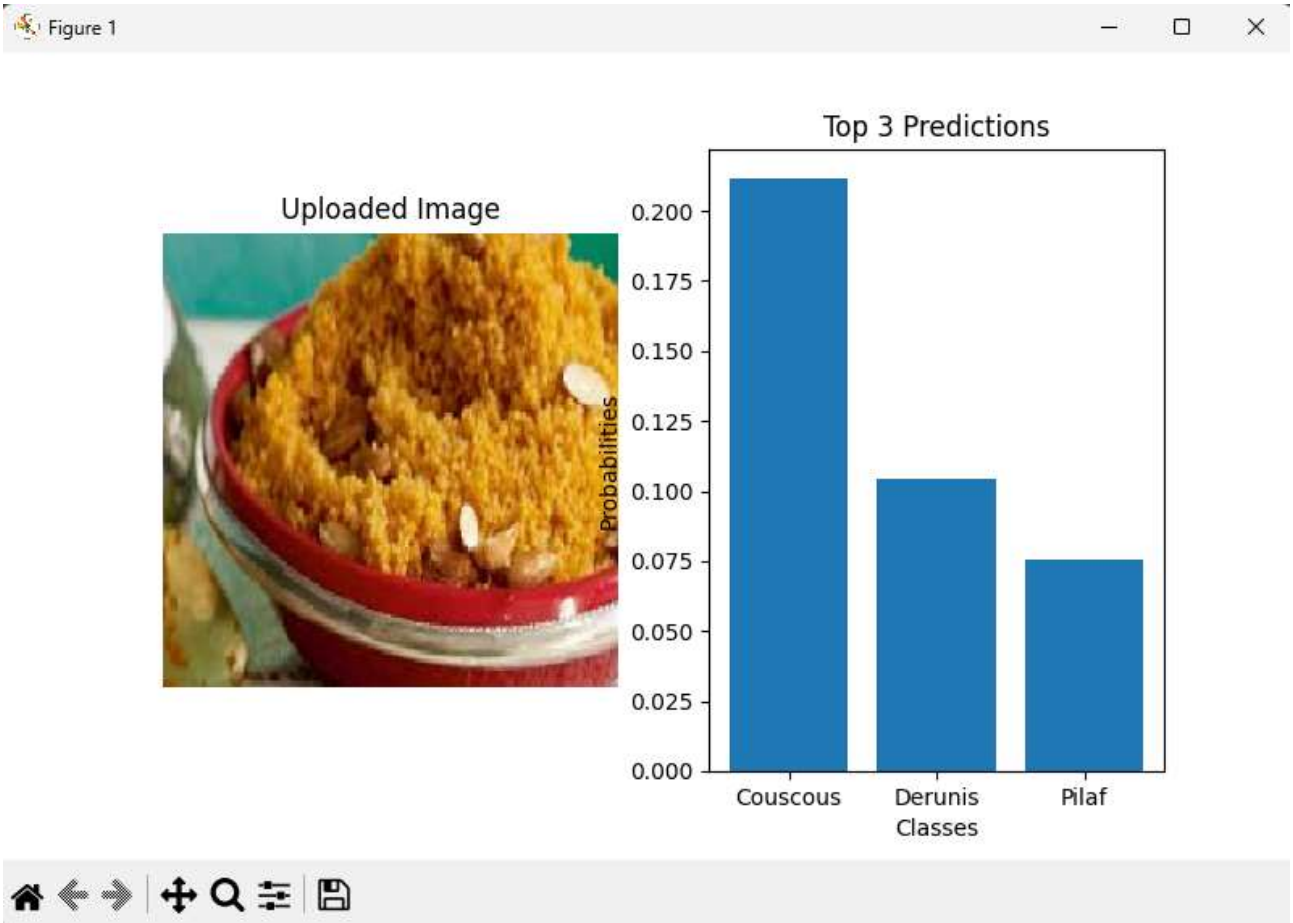


Рисунок 2.14 – Аналіз страви та виведення правильного результату

Нейронна мережа також може розпізнавати об'єкти на фото з дуже високою точністю. Наприклад, у випадку, наведеному вище, мережа правильно ідентифікувала «Кускус» на зображенні з 20% впевненістю, що вдвічі більше, ніж для наступного варіанту в списку.

Ці приклади демонструє, що, незважаючи на можливі помилки в розпізнаванні через схожість страв або низьку якість зображень, нейронні мережі здатні досягати високої точності при ідентифікації об'єктів, які мають чіткі та характерні ознаки. Яблуко має добре виражені візуальні особливості, такі як форма, колір і текстура, що допомагає нейронній мережі точно визначити його на фото.

2.4 Розробка програмного додатку

Розробка Telegram–бота розпочалася зі створення бази даних, яка служить для збереження інформації про користувачів, страви та їх калорійність, а також для детальної фіксації спожитої користувачами їжі та відповідної кількості калорій. Для реалізації цієї бази даних використовувалася бібліотека SQLite, що забезпечує ефективне управління даними в компактному форматі.

Схема бази даних, яка відображена на рисунку 2.15, включає кілька основних таблиць. Таблиця користувачів зберігає дані про кожного користувача бота, такі як їх унікальні ідентифікатори та ім’я. Таблиця страв містить перелік доступних страв з їх калорійністю. Також, є таблиця, в якій зберігається детальна інформація про щоденний раціон користувачів, де фіксуються спожиті страви з підраховуванням загальної кількості спожитих калорій та таблицю з загальної інформацією про кількість спожитих калорій на кожен день.

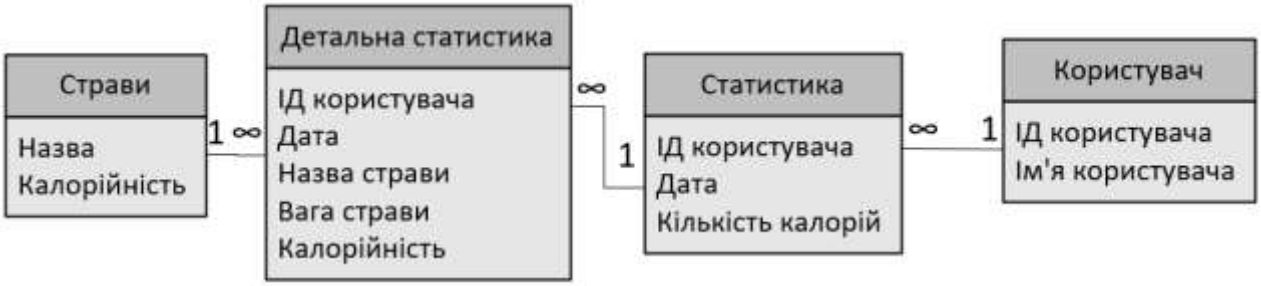


Рисунок 2.15 – Схема бази даних для Telegram-бота

Створення такої бази даних є критично важливим кроком у процесі розробки Telegram–бота, оскільки дозволяє забезпечити точний облік калорій та допомагає користувачам слідкувати за своїм харчуванням. SQLite була обрана для цієї мети завдяки своїй легкості у використанні, надійності та достатній продуктивності для обробки даних у рамках такого додатку.

При запуску Telegram–бота за допомогою команди /start, бот відреагує, вивівши вітальне повідомлення, зображене на рисунку 2.16.

Рисунок 2.16 – Виклик команди старт

При натисканні на кнопку «Інформація» бот відреагує, надаючи детальну інструкцію щодо користування Telegram-ботом та список команд, які допоможуть у роботі з ним, як зображено на рисунку 2.17.

Рисунок 2.17 – Детальна інформація про користування Telegram-ботом

Після перегляду інструкції користувач може надіслати зображення будь-якої страви. Telegram-бот, використовуючи потужність нейронних мереж, аналізує зображення, визначає назву страви та кількість калорій. Приклад такого аналізу зображений на рисунку 2.18.

Рисунок 2.18 – Аналіз зображення за допомого нейронної мережі та виведення результатів

Коли було відправлено зображення користувач може обрати ввести вагу страви для розрахунку калорійності та дізнатися скільки калорій він спожив, приклад представлено на рисунку 2.19.

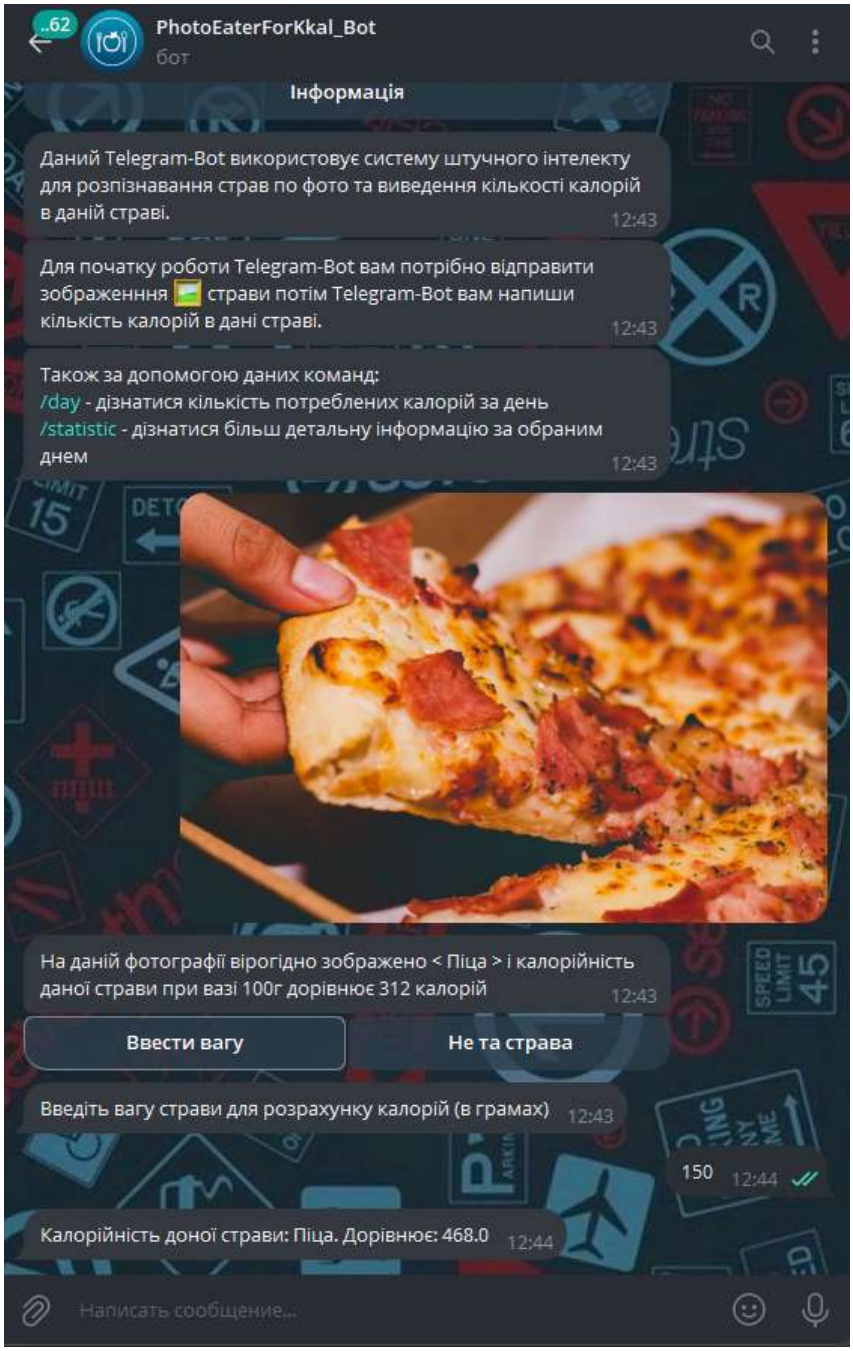


Рисунок 2.19 – Розрахунок калорійності страви

Також якщо нейронна мережа некоректно розпізнала страву користувач може обрати пункт «Не та страва» після якої користувачу виведе список страв з яких він може обрати ту страву яка була зображена на фото, представлено на рисунку 2.20.

Рисунок 2.20 – Вибір страви зі списку

Коли користувач вибере страву зі списку, бот запропонує йому ввести вагу цієї страви. Це необхідно для того, щоб розрахувати калорійність обраної страви, представлено на рисунку 2.19.

Якщо користувач не знайшов страву у списку, він може вибрати пункт меню «Немає в списку». Після цього йому буде запропоновано ввести назву страви, зображеної на фотографії, представленою на рисунку 2.21. Оскільки нейронна мережа наразі перебуває на етапі навчання, вона ще не знає всі можливі страви. Завдяки введенню нових назв страв користувачами, нейронна мережа зможе поступово розвиватися та розширювати свою базу даних страв. Це допоможе в майбутньому більш точно визначати калорійність для ширшого

спектра страв, забезпечуючи користувачам більш зручний і точний інструмент для контролю харчування.

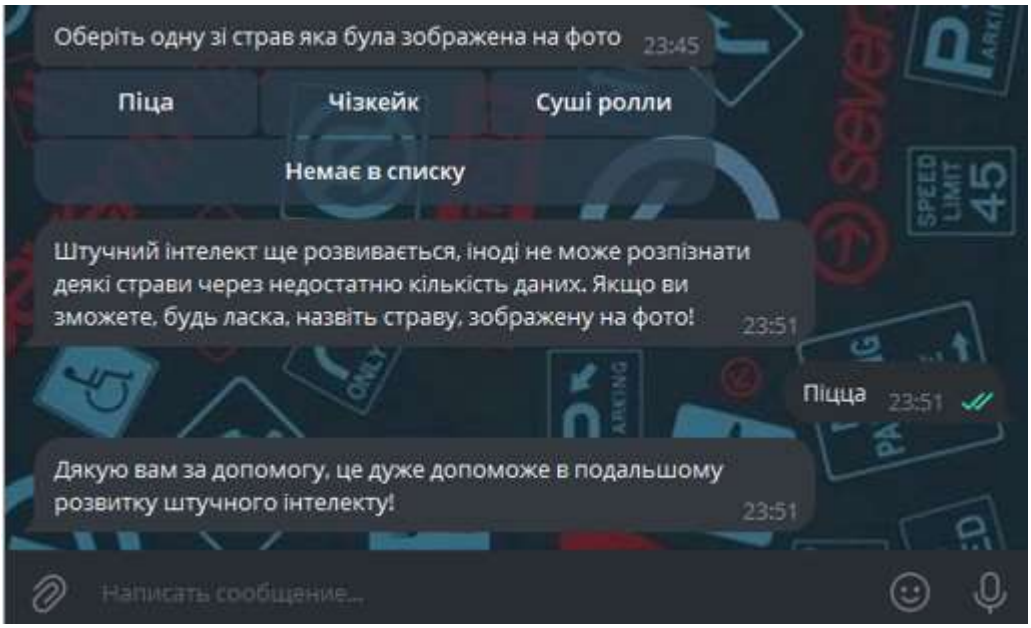


Рисунок 2.21 – Введення назви страви для розширення бази даних

Після того, як користувач ввів вагу страви та дізнався, скільки калорій він спожив, Telegram-бот зберігає цю інформацію в базу даних. Згодом користувач може скористатися командою /day, щоб отримати дані про кількість калорій, спожитих ним протягом дня. Приклад використання цієї команди представлений на рисунку 2.22.

Рисунок 2.22 – Виведення інформації про спожиті калорій по кожному дню

Користувач також може отримати більш детальну інформацію про спожиті страви та їхню калорійність за допомогою команди /statistic. Вибравши певну дату, користувач отримає від Telegram-бота детальний звіт про всі спожиті страви та їхню калорійність за цей день. Приклад використання цієї

команди показано на рисунку 2.23. Це дозволяє користувачам ретельніше аналізувати свій раціон, отримуючи розгорнуту інформацію про споживання калорій в будь-який конкретний день.

Рисунок 2.23 – Виведення детальної статистики за обраним днем

Розроблений Telegram-бот створений для полегшення користувачам завдання стеження за своїм раціоном. Завдяки інтеграції нейронної мережі, яка розпізнає страви за фотографіями, користувачі можуть легко знаходити інформацію про страви та їх калорійність із великої бази даних. Цей бот дозволяє відслідковувати спожиті калорії, контролювати щоденний раціон та зберігати історію харчування. Користувачі можуть просто завантажити фото

своєї їжі, і бот автоматично визначить, що це за страва, та виведе калорійність страви. Це робить процес управління харчуванням значно зручнішим та ефективнішим.

Аналізуючи статистичні дані за певну дату, можна скласти детальну статистику за тиждень. Таке дослідження дозволяє виявити тенденції та закономірності, що можуть бути непомітними при розгляді лише одноденних даних. Зведені результати аналізу за тиждень представлені у таблиці 2.7. Для кращої наочності та зручності сприйняття, дані з таблиці було візуалізовано у вигляді графіка, що наведений на рисунку 2.24.

Таблиця 2.7

Аналіз калорійності за тиждень

Дата	**Кількість калорій**
15.05.2022	873,2
16.05.2022	1916,5
17.05.2022	1079,3
18.05.2022	1300,7
19.05.2022	1321,5
20.05.2022	893,4
21.05.2022	1703,4

Рисунок 2.24 – Графік калорійності за тиждень

Завдяки цьому Telegram-боту користувач має можливість аналізувати калорійність свого раціону протягом певного періоду часу. Використовуючи бот, можна відстежувати, як змінювався раціон та кількість споживаних калорій щодня, щотижня або за будь-який інший обраний проміжок часу.

Цей інструмент не тільки допомагає вести облік калорій, але й надає візуалізовані звіти, що дозволяють легко оцінити прогрес у дотриманні дієти або плану харчування. Користувач може побачити, які продукти або звички найбільше впливають на загальну калорійність його раціону, що сприяє кращому розумінню своїх харчових звичок та їх впливу на здоров'я.

ВИСНОВОК

У даній кваліфікаційній роботі було успішно розроблено Telegram-бота з використанням нейронної мережі, який допомагає користувачам відслідковувати свій раціон харчування. Основною метою даного проекту було створення зручного інструменту для автоматизації розпізнавання страви за фотографією та надавати інформацію про їх калорійність.

Під час розробки було виконано такі завдання а саме:

– аналіз та вибір нейронної мережі і для цього було проведено дослідження сучасних методів розпізнавання зображень, також було обрано відповідну архітектуру нейронної мережі та навчено її на великому наборі даних зображень страв;

– другим завдання було інтеграцією нейронної мережі з Telegram-ботом і було створено Telegram-бота, який взаємодіє з користувачами, приймає фотографії страв та передає їх до нейронної мережі для обробки;

– далі було розроблено базу даних страв та їх калорійність і для цього було зібрано та структуровано інформацію про різноманітні страви та їх калорійний склад, що дозволяє надавати точні дані користувачам;

– також було проведено тестування та оптимізацію системи для цього проведено комплексне тестування бота на різних типах зображень для забезпечення високої точності розпізнавання та швидкості обробки.

Результати проекту демонструють, що запропонований Telegram-бот є ефективним інструментом для користувачів, які прагнуть контролювати свій раціон харчування. Завдяки можливості автоматичного розпізнавання страв за фотографіями, бот значно спрощує процес відслідковування споживаних калорій та допомагає користувачам приймати обґрунтовані рішення щодо свого харчування.

Подальший розвиток проекту може включати розширення функціоналу бота, зокрема додавання рекомендацій щодо здорового харчування, інтеграцію

з іншими платформами для обміну даними про раціон та підвищення точності розпізнавання за рахунок вдосконалення нейронної мережі. Таким чином, розроблений Telegram-бот має значний потенціал для подальшого розвитку та може стати важливим інструментом для підтримки здорового способу життя серед широкої аудиторії користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Огляд MyFitnessPal. Чи варто завантажити додаток MyFitnessPal URL: https://icoola.ua/blog/my-fitness-pal-app/ (дата звернення: 27.05.2024).

2. Bitesnap. URL: https://www.producthunt.com/products/bitesnap (дата звернення: 27.05.2024).

3. A Comprehensive Guide to Convolutional Neural Networks – the ELI5 way. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (дата звернення: 01.05.2024)

4. Convolutional neural network. URL: https://en.wikipedia.org/wiki/Convolutional_neural_network (дата звернення: 02.05.2024)

5. CNN | Introduction to Pooling Layer. URL: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/ (дата звернення: 02.05.2024)

6. Fully Connected Layer vs. Convolutional Layer: Explained. URL: https://builtin.com/machine-learning/fully-connected-layer (дата звернення: 03.05.2024)

7. MobileNet. URL: https://builtin.com/machine-learning/mobilenet (дата звернення: 13.05.2024)

8. What is MobileNetV2? Features, Architecture, Application and More. URL: https://www.analyticsvidhya.com/blog/2023/12/what-is-mobilenetv2/ (дата звернення: 13.05.2024)

9. ResNet: Residual Network URL: https://www.javatpoint.com/resnet-residual-network (дата звернення: 14.05.2024)

10. Searching for MobileNetV3 URL: https://arxiv.org/abs/1905.02244 (дата звернення: 15.05.2024)

11. I (VII) міжнародна науково-практична конференція здобувачів вищої освіти і молодих учених «Інформаційні технології: теорія і практика». Тези

доповідей (Дніпро 20 – 22 березня 2024 URL: https://ir.nmu.org.ua/handle/123456789/166565 (дата звернення: 02.05.2024)

ДОДАТОК А. Відомість матеріалів кваліфікаційної роботи

№ з/п	Позначення	Найменування	Кількість аркушів	Примітки
1				
2		Документація		
3				
4	САУ.КР.YY.ZZ.ПЗ	Пояснювальна записка	73	Формат А4
5				
6		Демонстраційний матеріал	N2	Презентація на CD-R
7				
8		Копія роботи	1	Диск CD-R
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				

					САУ.КР.YY.ZZ.ДА.ПЗ.			
Змін.	Аркуш	№ докум.	Підпис	Дата				
Розроб.		Горбенко М.М.			**Матеріали кваліфікаційної роботи**	Літ.	Аркуш	Аркушів
К. розд.		Хабарлак К.С.						
Керівн.		Хабарлак К.С.				НТУ «ДП», 12; 124-21ск-1		
Н.контр.		Хом’як Т.В.						
Зав. каф.		Желдак Т.А.						

ДОДАТОК Б КОД ПРОГРАМНОГО ПРОДУКТУ

Лістинг Б.1 – Код програми для створення Telegram-бота

```
import os
import sqlite3
import datetime

import cv2
import pandas as pd
import numpy as np
import requests
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Flatten, Input,
Conv2D, GlobalAveragePooling2D, Dropout, BatchNormalization
from tensorflow.keras.preprocessing.image import
img_to_array, load_img
import matplotlib.pyplot as plt
from tensorflow.keras.regularizers import l2
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.mobilenet_v2 import
preprocess_input, decode_predictions
from tensorflow.keras.models import load_model, Model

import telebot
from telebot import types

#Дані для підкбчення до тегелграм бота
bot_token = '6908768140:AAGmnkTQO7QI7dX9ndFzpAxF_KZo0l4F5zI'
save_path = 'Telegram_Image/'
bot =
telebot.TeleBot('6908768140:AAGmnkTQO7QI7dX9ndFzpAxF_KZo0l4F5zI')
os.makedirs(save_path, exist_ok=True)

#Підключення до баз даних
con_u = sqlite3.connect('users.db', check_same_thread=False)
con_f = sqlite3.connect('food.db', check_same_thread=False)
con_s = sqlite3.connect('statistic.db',
check_same_thread=False)
con_d = sqlite3.connect('detail_statistic.db',
check_same_thread=False)

label_to_class = {}
label_to_class_uk = {}
```

```
    top3_labels = []
    top3_labels_uk = []

    def load_classes():
        # Загрузка списка классов
        classes_path = 'diplomClasses.txt'
        with open(classes_path, 'r') as file:
            all_classes = file.read().splitlines()

        # Создание словаря для сопоставления классов с метками
для всех классов
        class_to_label_all = {class_name: i for i, class_name in
enumerate(all_classes)}
        print(all_classes)
        print(class_to_label_all)
        global label_to_class
        label_to_class    =    {v:    k    for    k,    v    in
class_to_label_all.items()}

    def load_classes_uk():
        # Загрузка списка классов
        classes_path = 'diplomClasses_Uk.txt'
        with open(classes_path, 'r') as file:
            all_classes = file.read().splitlines()

        # Создание словаря для сопоставления классов с метками
для всех классов
        class_to_label_all = {class_name: i for i, class_name in
enumerate(all_classes)}
        print(all_classes)
        print(class_to_label_all)
        global label_to_class_uk
        label_to_class_uk    =    {v:    k    for    k,    v    in
class_to_label_all.items()}

    cnn = tf.keras.models.load_model('MealRecogniser')
    name_photo = ""
    filename = 'new_image.txt'

    def image_AI(file_name):

        # Загрузка изображения
        img_path = "Telegram_Image/" + file_name
        img = image.load_img(img_path, target_size=(224, 224))

        # Преобразование изображения в массив numpy
        img_array = image.img_to_array(img)

        # Расширение размерности массива (добавление размерности
пакета)
        img_array = np.expand_dims(img_array, axis=0)
```

```
        # Предобработка изображения для модели MobileNetV2
(замените на свою модель)
        img_array = preprocess_input(img_array)

        # Получение предсказания для загруженного изображения
        prediction = cnn.predict(img_array)[0]

        # Отображение предсказания
        sorted_indices = np.argsort(prediction)[::-1]
        top3_indices = sorted_indices[:3]
        # Хранится название класа
        global label_to_class
        global label_to_class_uk
        top3_labels.clear()
        top3_labels_uk.clear()
        for idx in top3_indices:
            top3_labels.append(label_to_class.get(idx))
            top3_labels_uk.append(label_to_class_uk.get(idx))

    #Старт бота
    @bot.message_handler(commands=['start'])
    def start_bot(message):
        mycursor = con_u.cursor()
        sql = "SELECT * FROM users WHERE id = ?"
        adr = (str(message.from_user.id),)
        mycursor.execute(sql, adr)
        myresult = mycursor.fetchall()
        if myresult is None or myresult == [] or myresult == ():
            mycursor = con_u.cursor()
            sql = "INSERT INTO users (id, name) VALUES (?,?)"
            val                                                =
(str(message.from_user.id),str(message.from_user.username),)
            mycursor.execute(sql, val)
            con_u.commit()
        markup = types.InlineKeyboardMarkup()
        button   =   types.InlineKeyboardButton(text="Інформація",
callback_data="information")
        markup.add(button)
        text_all = "Вас вітає Telegram-Bot PhotoEaterForKkal. За
допомогою даного бота ви зможете слідкувати за свої раціоном. Для
більш детальної інформації натисніть кнопку 'Інформація'."
        with open('Logo (3).png', 'rb') as photo_file:
            bot.send_photo(message.from_user.id,
photo=photo_file)
        bot.send_message(message.from_user.id,           text_all,
reply_markup=markup)

    @bot.callback_query_handler(func=lambda call: True)
    def callback_query(call):
        if call.data == "information":
```

```
            text_1 = "Даний Telegram-Bot використовує систему
штучного інтелекту для розпізнавання страв по фото та виведення
кількості калорій в даній страві."
            text_2 = "Для початку роботи Telegram-Bot вам
потрібно відправити зображенння 🖼 страви потім Telegram-Bot вам
напиши кількість калорій в дані страві."
            text_3 = "Також за допомогою даних команд: \n/day -
дізнатися кількість потреблених калорій за день \n/statistic -
дізнатися більш детальну інформацію за обраним днем"
            bot.send_message(call.from_user.id, text_1)
            bot.send_message(call.from_user.id, text_2)
            bot.send_message(call.from_user.id, text_3)
        elif call.data == "button_end":
            text_end = "Штучний інтелект ще розвивається, іноді
не може розпізнати деякі страви через недостатню кількість даних.
Якщо ви зможете, будь ласка, назвіть страву, зображену на фото!"
            bot.send_message(call.from_user.id, text_end)
            bot.register_next_step_handler(call.message,
add_new_food)
        elif call.data == "button_food_1":
            text = "Введіть вагу страви для розрахунку калорій (в
грамах)"
            bot.send_message(call.from_user.id, text)
            bot.register_next_step_handler(call.message,
first_food_kkal)
        elif call.data == "button_food_2":
            text = "Введіть вагу страви для розрахунку калорій (в
грамах)"
            bot.send_message(call.from_user.id, text)
            bot.register_next_step_handler(call.message,
second_food_kkal)
        elif call.data == "button_food_3":
            text = "Введіть вагу страви для розрахунку калорій (в
грамах)"
            bot.send_message(call.from_user.id, text)
            bot.register_next_step_handler(call.message,
tree_food_kkal)
        elif call.data == "readmore":
            text = "Докладніше"
            bot.send_message(call.from_user.id, text)
            buttons_food_name(call.message)
        elif call.data == "ok":
            text = "Введіть вагу страви для розрахунку калорій (в
грамах)"
            bot.send_message(call.from_user.id, text)
            bot.register_next_step_handler(call.message,
first_food_kkal)
        elif call.data.startswith('calendar'):
            year,       month,       day       =       map(int,
call.data.split(':')[1].split('-'))
            selected_date = datetime.date(year, month, day)
```

```
            bot.send_message(call.message.chat.id,   f"Ви   обрали
дату: {selected_date.strftime('%d.%m.%Y')}")
            cursor = con_d.cursor()
            sql = "SELECT * FROM detail_statistic WHERE data = ?"
            val = (str(selected_date), )
            cursor.execute(sql, val)
            rows = cursor.fetchall()
            bot.send_message(call.message.chat.id,      f"Детальна
статистики на обрану дату:")
            for row in rows:
                bot.send_message(call.from_user.id,"Назва  страви:
" + row[2]+ ". Вага страви: " + row[3]+ ". Калорійність страви: "
+ row[4])


    #Завантаження фото з телеграму
    @bot.message_handler(content_types=['photo'])
    def hendle_photo(message):
        chat_id = message.chat.id
        file_info = bot.get_file(message.photo[-1].file_id)
        download_photo = bot.download_file(file_info.file_path)
        file_name = f'photo_{message.message_id}.jpg'
        file_path = os.path.join(save_path, file_name)
        with open(file_path, 'wb') as photo_file:
            photo_file.write(download_photo)
        global name_photo
        name_photo = file_name
        image_AI(file_name)
        first_step_food(chat_id)

    def first_step_food(chat_id):
        mycursor = con_f.cursor()
        sql = "SELECT kkal FROM food WHERE name_strava = ?"
        adr = (str(top3_labels[0]),)
        mycursor.execute(sql, adr)
        result = mycursor.fetchone()
        kkal_food = int(result[0])
        markup = types.InlineKeyboardMarkup(row_width=2)
        button_1 = types.InlineKeyboardButton(text="Ввести вагу",
callback_data="ok")
        button_2    =    types.InlineKeyboardButton(text="Не    та
страва", callback_data="readmore")
        markup.add(button_1, button_2)
        bot.send_message(chat_id, f"На даній фотографії вірогідно
зображено < {top3_labels_uk[0]} > і калорійність даної страви при
вазі 100г дорівнює {str(kkal_food)} калорій", reply_markup=markup)

    def buttons_food_name(message):
        print("1")
        markup = types.InlineKeyboardMarkup(row_width=3)
```

```
        button_1 =
types.InlineKeyboardButton(text=str(top3_labels_uk[0]),
callback_data="button_food_1")
        button_2 =
types.InlineKeyboardButton(text=str(top3_labels_uk[1]),
callback_data="button_food_2")
        button_3 =
types.InlineKeyboardButton(text=str(top3_labels_uk[2]),
callback_data="button_food_3")
        button_end = types.InlineKeyboardButton(text="Немає в
списку", callback_data="button_end")
        markup.add(button_1, button_2, button_3)
        markup.add(button_end)
        bot.send_message(message.chat.id, "Оберіть одну зі страв
яка була зображена на фото", reply_markup=markup)

    def add_new_food(message):
        global name_photo
        name_food = "," + message.text
        name_photo += name_food
        print(name_photo)
        with open(filename, 'a') as file:
            file.write(name_photo + '\n')
        text = "Дякую вам за допомогу, це дуже допоможе в
подальшому розвитку штучного інтелекту!"
        bot.send_message(message.from_user.id, text)

    def first_food_kkal(message):
        mycursor = con_f.cursor()
        sql = "SELECT kkal FROM food WHERE name_strava = ?"
        adr = (str(top3_labels[0]),)
        mycursor.execute(sql, adr)
        result = mycursor.fetchone()
        kkal_food = int(result[0])
        kkal = (kkal_food) * int(message.text)/100
        mycursor = con_s.cursor()
        sql = "SELECT * FROM statistic WHERE data = ? and id = ?"
        adr = (str(datetime.date.today()),
str(message.from_user.id),)
        mycursor.execute(sql, adr)
        myresult = mycursor.fetchall()
        print(myresult)
        if myresult is None or myresult == [] or myresult == ():
            mycur = con_s.cursor()
            sql_add = "INSERT INTO statistic (id, data, kkal_day)
VALUES (?,?,?)"
            val = (str(message.from_user.id),
str(datetime.date.today()), str(kkal), )
            mycur.execute(sql_add, val)
            con_s.commit()
        else:
```

```
            mycur = con_s.cursor()
            old_calories = float(myresult[0][2])
            new_calories = old_calories+kkal
            sql_update = "UPDATE statistic SET kkal_day = ? WHERE
data = ?"
            vals               =               (str(new_calories),
str(datetime.date.today()), )
            mycur.execute(sql_update, vals)
            con_s.commit()
        cursor = con_d.cursor()
        sql_add_deteil = "INSERT INTO detail_statistic (id, data,
name_strava, weight, kkal) VALUES (?,?,?,?,?)"
        vali = (message.from_user.id, str(datetime.date.today()),
str(top3_labels_uk[0]), message.text, str(kkal),)
        cursor.execute(sql_add_deteil, vali)
        con_d.commit()
        text_kkal     =     "Калорійність     доної     страви:
"+top3_labels_uk[0]+". Дорівнює: "+ str(kkal)
        bot.send_message(message.from_user.id, text_kkal)


    def second_food_kkal(message):
        mycursor = con_f.cursor()
        sql = "SELECT kkal FROM food WHERE name_strava = ?"
        adr = (str(top3_labels[1]),)
        mycursor.execute(sql, adr)
        result = mycursor.fetchone()
        kkal_food = int(result[0])
        kkal = (kkal_food)*int(message.text)/100
        mycursor = con_s.cursor()
        sql = "SELECT * FROM statistic WHERE data = ? and id = ?"
        adr             =             (str(datetime.date.today()),
str(message.from_user.id),)
        mycursor.execute(sql, adr)
        myresult = mycursor.fetchall()
        print(myresult)
        if myresult is None or myresult == [] or myresult == ():
            mycur = con_s.cursor()
            sql_add = "INSERT INTO statistic (id, data, kkal_day)
VALUES (?,?,?)"
            val           =           (str(message.from_user.id),
str(datetime.date.today()), str(kkal),)
            mycur.execute(sql_add, val)
            con_s.commit()
        else:
            mycur = con_s.cursor()
            old_calories = float(myresult[0][2])
            new_calories = old_calories + kkal
            sql_update = "UPDATE statistic SET kkal_day = ? WHERE
data = ?"
```

```
            vals                =                (str(new_calories),
str(datetime.date.today()),)
            mycur.execute(sql_update, vals)
            con_s.commit()
        cursor = con_d.cursor()
        sql_add_deteil = "INSERT INTO detail_statistic (id, data,
name_strava, weight, kkal) VALUES (?,?,?,?,?)"
        vali = (message.from_user.id, str(datetime.date.today()),
str(top3_labels_uk[1]), message.text, str(kkal),)
        cursor.execute(sql_add_deteil, vali)
        con_d.commit()
        text_kkal   =   "Калорійність   доної   страви:   "   +
top3_labels_uk[1] + ". Дорівнює: " + str(kkal)
        bot.send_message(message.from_user.id, text_kkal)

    def tree_food_kkal(message):
        mycursor = con_f.cursor()
        sql = "SELECT kkal FROM food WHERE name_strava = ?"
        adr = (str(top3_labels[2]),)
        mycursor.execute(sql, adr)
        result = mycursor.fetchone()
        kkal_food = int(result[0])
        kkal = (kkal_food)*int(message.text)/100
        mycursor = con_s.cursor()
        sql = "SELECT * FROM statistic WHERE data = ? and id = ?"
        adr             =             (str(datetime.date.today()),
str(message.from_user.id), )
        mycursor.execute(sql, adr)
        myresult = mycursor.fetchall()
        print(myresult)
        if myresult is None or myresult == [] or myresult == ():
            mycur = con_s.cursor()
            sql_add = "INSERT INTO statistic (id, data, kkal_day)
VALUES (?,?,?)"
            val           =           (str(message.from_user.id),
str(datetime.date.today()), str(kkal),)
            mycur.execute(sql_add, val)
            con_s.commit()
        else:
            mycur = con_s.cursor()
            old_calories = float(myresult[0][2])
            new_calories = old_calories + kkal
            sql_update = "UPDATE statistic SET kkal_day = ? WHERE
data = ?"
            vals                =                (str(new_calories),
str(datetime.date.today()),)
            mycur.execute(sql_update, vals)
            con_s.commit()
        cursor = con_d.cursor()
        sql_add_deteil = "INSERT INTO detail_statistic (id, data,
name_strava, weight, kkal) VALUES (?,?,?,?,?)"
```

```
        vali = (message.from_user.id, str(datetime.date.today()),
str(top3_labels_uk[2]), message.text, str(kkal),)
        cursor.execute(sql_add_deteil, vali)
        con_d.commit()
        text_kkal = "Калорійність доної страви: " +
top3_labels_uk[2] + ". Дорівнює: " + str(kkal)
        bot.send_message(message.from_user.id, text_kkal)


    @bot.message_handler(commands=['day'])
    def start_bot(message):
        cursor = con_s.cursor()
        sql = "SELECT data, kkal_day FROM statistic WHERE id = ?"
        val = (str(message.from_user.id),)
        cursor.execute(sql, val)
        rows = cursor.fetchall()
        for row in rows:
            bot.send_message(message.from_user.id,          "Дата:
"+row[0]+". Кількість калорій: " + row[1])

    def create_calendar(year, month):
        markup = types.InlineKeyboardMarkup(row_width=7)
        now = datetime.datetime.now()
        current_year = now.year
        current_month = now.month
        ukrainian_months = {
            1: "Січень",
            2: "Лютий",
            3: "Березень",
            4: "Квітень",
            5: "Травень",
            6: "Червень",
            7: "Липень",
            8: "Серпень",
            9: "Вересень",
            10: "Жовтень",
            11: "Листопад",
            12: "Грудень"
        }
        month_name = ukrainian_months[current_month]
        markup.add(types.InlineKeyboardButton(str(month_name),
callback_data="ignore"))
        # Создание заголовка с названиями дней недели
        markup.add(types.InlineKeyboardButton("Пн",
callback_data="ignore"),
                   types.InlineKeyboardButton("Вт",
callback_data="ignore"),
                   types.InlineKeyboardButton("Ср",
callback_data="ignore"),
                   types.InlineKeyboardButton("Чт",
callback_data="ignore"),
```

```
                    types.InlineKeyboardButton("Пт",
callback_data="ignore"),
                    types.InlineKeyboardButton("Сб",
callback_data="ignore"),
                    types.InlineKeyboardButton("Вс",
callback_data="ignore"))
        # Получение дня недели для первого дня месяца
        first_weekday = datetime.date(year, month, 1).weekday()

        # Пустые кнопки для "заполнения" места до начала месяца
        #for _ in range(first_weekday):
        #markup.add(types.InlineKeyboardButton(str(month_name),
callback_data="ignore"))
        # Заполнение календаря днями месяца
        day = 1
        while day <= 31:    # Цикл до 31, чтобы учесть все
возможные дни месяца
            row = []
            for i in range(7):
                if (day <= 31) and (month == current_month or
year == current_year):

row.append(types.InlineKeyboardButton(str(day),
callback_data=f"calendar:{year}-{month}-{day}"))
                else:
                    row.append(types.InlineKeyboardButton("     ",
callback_data="ignore"))
                day += 1
            markup.add(*row)
        return markup

    @bot.message_handler(commands=['statistic'])
    def statistic(message):
        bot.send_message(message.chat.id, "Выберите дату в цьому
місяці:",
reply_markup=create_calendar(datetime.datetime.now().year,
datetime.datetime.now().month))



    if __name__ == "__main__":
        load_classes()
        load_classes_uk()
        bot.polling()
```

Лістинг Б.2 – Код для навчання нейронної мережі

```
import os
import cv2
import pandas as pd
```

```
import numpy as np
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Flatten, Input, Conv2D, GlobalAveragePooling2D, Dropout, BatchNormalization
from tensorflow.keras.preprocessing.image import img_to_array, load_img
import matplotlib.pyplot as plt
from tensorflow.keras.regularizers import l2
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input, decode_predictions
from tensorflow.keras.models import load_model, Model

classes_path = 'diplomClasses.txt'
with open(classes_path, 'r') as file:
    all_classes = file.read().splitlines()

# Создание словаря для сопоставления классов с метками для всех классов
class_to_label_all = {class_name: i for i, class_name in enumerate(all_classes)}
print(all_classes)
print(class_to_label_all)
label_to_class = {v: k for k, v in class_to_label_all.items()}

# Загрузка данных из CSV-файла
csv_path = 'diplomImages.csv'
data = pd.read_csv(csv_path)

# Папка с изображениями
image_folder = 'DiplomImages'

# Преобразование меток классов в числовые значения
print(data)
# Разделение данных на обучающий и тестовый наборы
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)
print(train_data)
print(test_data)

#Загрузка та подгатовка фотографий
def load_and_preprocess_image(image_path, label):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_image(image, channels=3)
```

```
        image = tf.image.resize(image, (224, 224))
        image_array = tf.cast(image, tf.float32) / 255.0
        return image_array, label

    train_images = []
    train_labels = []

    for index, row in train_data.iterrows():
        img_path = os.path.join(image_folder,
f"{row['PhotoId']}")
        print(img_path)
        img, label = load_and_preprocess_image(img_path,
row['Name'])
        train_images.append(img)
        train_labels.append(class_to_label_all[row['Name']])
        print('added')

    test_images = []
    test_labels = []

    for index, row in test_data.iterrows():
        img_path = os.path.join(image_folder,
f"{row['PhotoId']}")
        print(img_path)
        img, label = load_and_preprocess_image(img_path,
row['Name'])
        test_images.append(img)
        test_labels.append(class_to_label_all[row['Name']])
        print('added')

    train_labels = to_categorical(train_labels, num_classes=48)
    test_labels = to_categorical(test_labels, num_classes=48)
    train_images = tf.stack(train_images)
    test_images = tf.stack(test_images)

    #Опитимизация та рандомизация DataSets
    AUTOTUNE = tf.data.AUTOTUNE

    train_dataset =
tf.data.Dataset.from_tensor_slices((train_images, train_labels))
    train_dataset = train_dataset.shuffle(1000)
    train_dataset = train_dataset.batch(64)
    train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)

    val_dataset =
tf.data.Dataset.from_tensor_slices((test_images, test_labels))
    val_dataset = val_dataset.shuffle(1000)
    val_dataset = val_dataset.batch(64)
    val_dataset = val_dataset.prefetch(buffer_size=AUTOTUNE)

    #Створення слою аугментации
```

```
    data_augmentation = tf.keras.Sequential([
      tf.keras.layers.RandomFlip('horizontal'),
      tf.keras.layers.RandomRotation(0.2),
    ])

    #Завантаження базовою модели та видключення ии навчання
    base_model = MobileNetV2(weights='imagenet',
include_top=False, input_shape=(224, 224, 3)) #назва нейронної
мережі
    base_model.trainable = False

    #Створення модели
    inputs = Input(shape=(224,224,3))
    x = data_augmentation(inputs)
    x = base_model(x)
    x = GlobalAveragePooling2D()(x)
    x = Flatten()(x)
    x = Dense(256, activation = 'relu',
kernel_regularizer=l2(0.01))(x)
    x = Dropout(0.5)(x)  # Добавление Dropout слоя
    x = Dense(48, activation = 'softmax',
kernel_regularizer=l2(0.01))(x)
    outputs = x
    cnn = keras.Model(inputs, outputs)
    cnn.compile(optimizer=tf.keras.optimizers.Adam(0.001),
                loss='categorical_crossentropy',
                metrics=['accuracy'])
    cnn.summary()

    #Обучение модели
    history = cnn.fit(train_dataset, epochs=10,
                      validation_data=val_dataset)

    # Plot training and validation loss
    plt.figure(figsize=(12, 6))

    # Plot training & validation loss values
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(['Train', 'Validation'], loc='upper right')

    # Plot training & validation accuracy values
    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
```

```
    plt.ylabel('Accuracy')
    plt.legend(['Train', 'Validation'], loc='lower right')

    plt.show()
    # Продолжить обучение с еще 5 эпох
    additional_epochs = 5
    history_continued           =           cnn.fit(train_dataset,
epochs=additional_epochs, validation_data=val_dataset)

    # Добавить новые эпохи к существующей истории
    history.history['loss'].extend(history_continued.history['los
s'])
    history.history['val_loss'].extend(history_continued.history[
'val_loss'])
    history.history['accuracy'].extend(history_continued.history[
'accuracy'])
    history.history['val_accuracy'].extend(history_continued.hist
ory['val_accuracy'])

    # Построить обновленные графики
    plt.figure(figsize=(12, 6))

    # Plot training & validation loss values
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(['Train', 'Validation'], loc='upper right')

    # Plot training & validation accuracy values
    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(['Train', 'Validation'], loc='lower right')

    plt.show()

    #Збереження модели
    #cnn.save('my_model.h5')
```

Лістинг Б.3 – Код для створення бази даних

```
    import sqlite3
    import csv
```

```
    #Створення таблиці статистика
    con_s = sqlite3.connect('statistic.db')
    cur_s = con_s.cursor()
    cur_s.execute('''CREATE TABLE statistic (id VARCHAR(255),
data VARCHAR(255), kkal_day VARCHAR(255)) ''')

    #Створення таблиці користувачі
    con_u = sqlite3.connect('users.db')
    cur_u = con_u.cursor()
    cur_u.execute('''CREATE TABLE users (id VARCHAR(255), name
VARCHAR(255))''')

    #Створення таблиці дательна статистика
    con_d = sqlite3.connect('detail_statistic.db')
    cur_d = con_d.cursor()
    cur_d.execute('''CREATE TABLE detail_statistic (id
VARCHAR(255), data VARCHAR(255), name_strava VARCHAR(255), weight
VARCHAR(255), kkal VARCHAR(255))''')

    #Створення таблиці страви
    con_f = sqlite3.connect('food.db')
    cur_f = con_f.cursor()
    cur_f.execute('''CREATE TABLE food (id VARCHAR(255),
name_strava VARCHAR(255), kkal VARCHAR(255))''')

    with open('diplomImages.csv', 'r', newline='') as file:
        # Создаем объект чтения CSV как словарь
        reader = csv.DictReader(file)
        i = 1
        # Проходимся по каждой строке в файле
        for row in reader:
            #Завантаження даних про страви до таблиці страви
            mycursor = con_f.cursor()
            sql = "INSERT INTO food (id, name_strava, kkal)
VALUES (?,?,?)"
            name_strava = {}
            for item in reader:
                group_name = item['Name']
                if group_name not in name_strava:
                    name_strava[group_name] = item['Cal']
            for key, value in name_strava.items():
                val = (i, key, value)
                mycursor.execute(sql, val)
                i = i + 1
            con_f.commit()
```

Відгук
на кваліфікаційну роботу бакалавра
студента(ки) групи *124 – 21ск – 1*
спеціальності *124 Системний аналіз*

Тема кваліфікаційної роботи: *«Розробка Telegram бота для розрахунку калорійності страв по фотографії з використанням штучного інтелекту»*

Обсяг кваліфікаційної роботи *73* стор.

Мета кваліфікаційної роботи: *розробка Telegram-бота, який може аналізувати фотографії страв, розпізнавати їх, визначати їх калорійність і надавати користувачам відповідну інформацію у зручному форматі та полегшити ведення статистики що до харчування*

Актуальність теми: *Telegram бот, який використовує нейронну мережу для розрахунку калорійності страв по фотографії, може стати зручним та доступним інструментом для людей, які хочуть контролювати своє харчування*.

Тема кваліфікаційної роботи безпосередньо пов'язана з об'єктом діяльності бакалавра спеціальності 124 Системний аналіз, оскільки *1) використано передові розробки в області аналізу даних та машинного навчання; 2) проведено аналіз сучасної літератури в області нейронних мереж, відмічено переваги та недоліки існуючих методів; 3) проаналізовано архітектури нейронних мереж та розроблено новий Telegram бот для оцінки калорійності страв за її зображенням.*

Виконані в кваліфікаційній роботі завдання відповідають вимогам ступеня бакалавра. Оригінальність наукових рішень полягає в *1) в розробці та впроваджені нового ефективного алгоритму розпізнавання типу страви за зображенням та оцінки її калорійності; 2) розробці нового програмного рішення, що складається з серверної частини з БД SQLite та нейронної мережі, клієнтської частини у вигляді Telegram бота, що завжди доступний користувачу.*

Практичне значення результатів кваліфікаційної роботи полягає в *можливості впровадження розробленого застосунку в компаніях з дієтології в Україні та поширенню результатів роботи на інші задачі класифікації в багатьох*

компаніях, що використовують нейронні мережі для аналізу та обробки зображень.

Висновки підтверджують можливість використання результатів роботи в *1) підприємствах, що розробляють та підтримують застосунки з моніторингу здорового ритму життя, дотримання дієт тощо; 2) отримані результати аналізу архітектур нейронних мереж можуть бути використані при вирішення інших задач класифікації зображень.*

Оформлення пояснювальної записки та демонстраційного матеріалу до неї виконано згідно з вимогами. Роботу виконано самостійно, відповідно до завдання та у повному обсязі.

У роботі відзначено такі недоліки: *не виявлено.*

Кваліфікаційна робота в цілому заслуговує оцінки: *відмінно.*

З урахуванням висловлених зауважень автор заслуговує присвоєння освітньої кваліфікації «бакалавр з системного аналізу».

Керівник кваліфікаційної роботи бакалавра,
доктор філософії, асистент каф. САУ _____/ Хабарлак К.С.

Рецензія
на кваліфікаційну роботу бакалавра
студента(ки) групи *124 – 21ск – 1*
спеціальності *124 Системний аналіз*

Тема кваліфікаційної роботи:
_____
_____
_____
Обсяг кваліфікаційної роботи: _____
Висновок про відповідність кваліфікаційної роботи завданню та освітньо-професійній програмі спеціальності _____
_____
_____
Загальна характеристика кваліфікаційної роботи, ступінь використання нормативно-методичної літератури та передового досвіду
_____
_____
_____
Позитивні сторони кваліфікаційної роботи:
_____
_____
_____

Основні недоліки кваліфікаційної роботи:
_____
_____
_____
_____
Кваліфікаційна робота в цілому заслуговує оцінки: ____

З урахуванням висловлених зауважень автор (не) заслуговує присвоєння освітньої кваліфікації «бакалавр з системного аналізу».

Рецензент,
науковий ступінь, вчене звання, посада _____ /
ПІБ