МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПОЛІСЬКИЙ НАЦІОНАЛЬНИ УНІВЕРСИТЕТ

Факультет інформаційних технологій, обліку та фінансів
Кафедра комп'ютерних технологій
і моделювання систем

Кваліфікаційна робота
на правах рукопису

Новік Владислав Сергійович
(прізвище, ім'я, по батькові здобувача освіти)

УДК 004.8:004.42

КВАЛІФІКАЦІЙНА РОБОТА

Інформаційна система розпізнавання хвороб рослин
(тема роботи)

126 «Інформаційні системи та технології»
(шифр і назва спеціальності)

Подається на здобуття освітнього ступеня бакалавр

кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____
(підпис, ініціали та прізвище здобувача вищої освіти)

Керівник роботи
Молодецька Катерина Валеріївна,
доктор технічних наук, професор

Житомир – 2024

Висновок кафедри ___

за результатами попереднього захисту: _____

Протокол засідання кафедри _____
№ _____від «_____» _____20_____р.

Завідувач кафедри _____

_____ _____ _____
(науковий ступінь, вчене звання) (підпис) (прізвище, ім'я, по батькові)

«_____» _____20_____р.

Результати захисту кваліфікаційної роботи

Здобувач вищої освіти _____захистив (ла)
(прізвище ,ім'я, по батькові)

кваліфікаційну роботу з оцінкою:

сума балів за 100-бальною шкалою _____
за шкалою ECTS _____
за національною шкалою _____

Секретар ЕК

_____ _____ _____
(науковий ступінь, вчене звання) (підпис) (прізвище, ім'я, по батькові)

АНОТАЦІЯ

Новік В.С. Інформаційна система розпізнавання хвороб рослин. – Кваліфікаційна робота на правах рукопису.

Кваліфікаційна робота присвячена розробленню інформаційної системи для автоматизованого розпізнавання хвороб рослин на основі аналізу зображень листових пластин з використанням технологій штучного інтелекту. Система базується на застосуванні згорткових нейронних мереж для задач класифікації зображень та розпізнавання захворювань за візуальними ознаками.

Робота вирішує актуальну проблему складності своєчасної та точної діагностики хвороб рослин традиційними методами, що вимагає залучення висококваліфікованих експертів та є ресурсозатратним процесом. Розроблена інформаційна система дозволяє значно підвищити ефективність виявлення захворювань та скоротити витрати в сільськогосподарській галузі.

В роботі описано процес підготовки навчальних даних, побудови та навчання згорткової нейронної мережі з використанням методів збагачення даних. Реалізовано Android-додаток з інтеграцією навченої моделі для розпізнавання хвороб в режимі реального часу. Система демонструє високу швидкодію та достатню точність класифікації.

Ключові слова: нейронна мережа, комп'ютерний зір, класифікація зображень, розпізнавання хвороб рослин, збагачення даних, Android.

SUMMARY

Novik V.S. Information System for Plant Disease Recognition. - Qualification work as a manuscript.

The qualification work is devoted to the development of an information system for automated recognition of plant diseases based on the analysis of leaf blade images using artificial intelligence technologies. The system is based on the application of convolutional neural networks for image classification tasks and disease recognition by visual features.

The work solves the urgent problem of the complexity of timely and accurate diagnosis of plant diseases by traditional methods, which requires the involvement of highly qualified experts and is a resource-intensive process. The developed information system significantly increases the efficiency of disease detection and reduces costs in the agricultural industry.

The work describes the process of preparing training data, building and training a convolutional neural network using data augmentation methods. An Android application with integration of the trained model for real-time disease recognition has been implemented. The results of evaluating the classification accuracy of the developed system are presented.

Keywords: neural network, computer vision, image classification, plant disease recognition, data augmentation, Android.

Зміст

ВСТУП

Хвороби рослин можуть мати значний вплив на врожайність сільськогосподарських культур і продовольчу безпеку. Традиційні методи розпізнавання хвороб рослин займають багато часу, а фермерам часто бракує знань та навичок, необхідних для їх виявлення.

Сучасний процес діагностування хвороб в рослин полягає в тому, що фермери або агрономи в ручну перевіряють кожну рослину, що займає багато часу та коштів. Саме тому актуальною є задача для розробки ефективних інструментів для моніторингу здоров’я рослин. Основною задачею є інтегрування комп’ютерного зору в цю систему для ідентифікації захворювання з високою швидкістю та точністю.

Мета: Підвищення ефективності діагностики та контролю за станом рослинності з використанням інформаційної системи розпізнавання хвороб рослин.

Предмет дослідження: моделі й методи розпізнавання хвороб рослин, що базуються використанні нейронних мереж для аналізу та класифікації зображень. Об'єкт дослідження: Процес розпізнавання та класифікації хвороб рослин за допомогою нейронних мереж на основі зображень, а також методи машинного навчання, що лежать в основі роботи інформаційної системи.

Перелік публікацій автора за темою дослідження:

1. Новік В.С. ОБГРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБЛЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМА РОЗПІЗНАВАННЯ ХВОРОБ РОСЛИН: матеріали міжфакультетної студентської науково-практичної конференції, м. Житомир, 14 листопада 2023 р. Житомир: ПНУ, 2023. С. 2-3
2. Новік В.С. ОБГРУНТУВАННЯ ВИБОРУ АРХІТЕКТУРИ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ РОЗРОБЛЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМА РОЗПІЗНАВАННЯ ХВОРОБ РОСЛИН: матеріали всеукраїнської студентської науково-практичної конференції, м. Житомир, 10 квітня 2024 р. Житомир: ПНУ, 2024. С. 1-3

Розділ 1. Аналіз підходів до створення технології розпізнавання хвороб рослин

1.1 Роль нейронних мереж в задачах комп'ютерного зору та класифікації зображень

Нейронні мережі відіграють ключову роль в задач комп’ютерного зору та класифікації зображень. Вони здатні навчатися на великій кількості інформації та знаходити певні особливості на зображеннях, що і робить їх корисними в завданнях з аналізом зображень. Нейронна мережа – це математична модель, яка намагається імітувати роботу людського мозку. Вона складається з вузлів, які обробляють та передають інформацію. Під час навчання на наборі даних, нейронна мережа підлаштовує себе так, щоб зменшити похибку між прогнозами та відомими даними.

Роль нейронних мереж в комп’ютерному зору та класифікації зображень полягає в:

- Класифікації зображень – визначення категорії об’єкта
- Сегментації зображень – розділення зображення на частини для аналізу окремих сегментів
- Відстеження об’єктів – відстежувати рух та позицію об’єкта на послідовності зображень або відео
- Виявлення об’єктів – локалізація та класифікація об’єкта

Також існує багато типів нейронних мереж для вирішення задач в комп'ютерному зорі та класифікації зображень, а саме згорткові (CNN), рекуретні (RNN), з довгою короткочасною пам’яттю (LSTM) та трансформери. Порівняльний аналіз типів нейронних мереж було проведено відповідно до таблиці 1.

Таблиця 1 – Порівняльний аналіз

Тип	Переваги	Недоліки
CNN	Ефективні для виявлення особливих характеристик на зображеннях	Не ефективні у вирішенні завдань, що потребують пам'яті
RNN	Здатність моделювати залежності в часі	Проблеми від зникаючого градієнту
LSTM	Можуть зберігати інформацію протягом тривалого часу	Складніші за RNN
Трансформери	Ефективні в задачах, що потребують механізму уваги	Вимагають багато обчислювальних ресурсів

Відповідно до табл. 1 для вирішення поставленої задачі найкраще підходить згорткова нейронна мережа, адже вона ефективна в задачах класифікації зображень.

1.2 Архітектури нейронних мереж для розпізнавання хвороб рослин

У рамках дослідження розглядається використання нейронних мереж для розпізнавання хвороб рослин за зображенням листової пластини. Одним із ключових завдань для вирішення даної проблеми є вибір архітектури для навчання нейронної мережі. Існують різноманітні архітектури від тих, що допомагають розпізнати особливості зображення до тих, що використовують зв’язки між шарами, які вирішують проблему зникнення градієнту. Використання правильно підібраної архітектури має декілька переваг, а саме:

- Ефективність використання ресурсів - деякі архітектури оптимізовані для роботи на мобільних пристроях, тобто архітектура дозволяє ефективно використовувати ресурси системи

- Швидкість - деякі архітектури дозволяють навчити модель значно швидше, що в подальшому дозволить отримувати результат класифікації миттєво
- Точність - деякі архітектури мають складнішу будову, що дозволяє їм показувати кращі результати для певних задач

Для розробки моделі для мобільного застосунку існує три найбільш популярні архітектури такі як:

- MobileNetV2 – архітектура розроблена спеціально для використання на мобільних пристроях. Мають низьку кількість параметрів та вимагає менше ресурсів.
- ShuffleNet – використовує блоки із змішуванням каналів та групову згортку для зменшення використання ресурсів.
- EfficientNet – сімейство архітектур, збалансовані точністю та використанням ресурсів.

Для вибору кращої моделі виконано порівняння існуючих архітектур відповідно до критеріїв, наведених в таблиці 2.

Таблиця 2 – Порівняння архітектур

Архітектура	Точність	Ефективність	Складність	FLOPS
MobileNetV2	Висока	Висока	Низька	Середня
ShuffleNet	Висока	Висока	Низька	Низька
EfficientNet	Дуже висока	Висока	Висока	Середня

Відповідно до результатів порівняння було обрано архітектуру MobileNetV2, адже вона ідеально підходить для вирішення поставленої задачі, а саме вона має високу точність, ефективно використовує ресурси системи, має низьку складність та невелику кількість операцій FLOPS. Також потрібно визначити потрібні бібліотеки для навчання нейронної мережі відповідно до рис. 1.1

```
import os

import numpy as np
import pandas as pd

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import callbacks, layers, Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Рисунок 1.1 – Бібліотеки для навчання

Бібілотека os потрібна для роботи з файловою системою. Numpy для роботи з масивами та матричними обчисленнями, pandas для роботи з структурованими даними, tensorflow основна бібліотека для машинного навчання та keras для швидкого створення та конфігурації моделі.

1.3 Деталізація нейронної мережі та параметрів навчання для розпізнавання хвороб рослин

Після обрання типу та архітектури нейронної мережі, потрібно обрати датасет для навчання, середу розробки та оптимальну конфігурацію для досягнення максимально можливої точності.

В якості мови програмування було обрано Python, адже він чудово працює з машинним навчанням. Також розробка буде проводитися в Jupyter Notebook, адже це дозволить нам запускати код по комірках, що є зручним варіантом, адже результати коду будуть одразу доступні.

Проаналізувавши датасети у відкритому доступі було обрано датасет “PlantVillage” він містить понад 54000 зображень рослин та 38 класів. Відповідно до рис. 1.2 датасет потрібно завантажити в середовище та визначити дані для тренування та валідації.

```
dataset_root = "F:/CNN"

train_dir = os.path.join(dataset_root, "train")
test_dir = os.path.join(dataset_root, "valid")
```

Рисунок 1.2 – Завантаження датасету

Для навчання нейронної мережі такоє використовується метод Data Augmentation, який змінює зображення за відповідними показниками відповідно до рис. 1.3.

```
train_aug = ImageDataGenerator(
    rescale=1/255.0,
    fill_mode="nearest",
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    shear_range=0.2,
)

train_data = train_aug.flow_from_directory(
    train_dir,
    target_size=(image_size, image_size),
    batch_size=batch_size,
    class_mode="categorical"
)
```

Рисунок 1.3 - Data Augmentation

Даний код змінює певні параметри зображення, а саме нормалізація значень пікселів, діапазон вертикальних та горизонтальних зсувів, масштабування та діапазон діагональних зсувів. Потім код застосовує дані трансформації до даних навчання.

Для використання даного методу потрібні бібліотеки “tensorflow” та “keras”. Спочатку інтегруємо базову модель “MobileNetV2”, не включаючи верхні шари, оскільки ми хочемо використовувати її для 38 класів і використовуємо попередньо навчені ваги для “ImageNet” відповідно до рис. 1.4.

```
mbnet_v2 = keras.applications.MobileNetV2(
    weights="imagenet",
    include_top=False,
    input_shape=input_shape
)

mbnet_v2.trainable = False
```

Рисунок 1.4 – Завантаження MobileNetV2

Тепер створюємо невелику висхідну модель поверх MobileNetV2, використовуючи функціональний API згідно рис. 1.5.

```
inputs = keras.Input(shape=input_shape)

x = mbnet_v2(inputs, training = False)

x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Dense(len(cats), activation="softmax")(x)

model = Model(inputs=inputs, outputs=x)

model.summary()
```

Рисунок 1.5 - Модель нейронної мережі

Далі потрібно налаштувати модель нейронної мережі та додати функцію компіляції та механізм раннього зупинення навчання відповідно до рис. 1.6.

```
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])

early_stopping_cb = callbacks.EarlyStopping(monitor="loss", patience=3)
```

Рисунок 1.6 – Функція компіляції

Тепер потрібно запустити процес навчання на навчальних даних відповідно до рис. 1.7.

```
epochs = 30

history = model.fit(
    train_data,
    epochs=epochs,
    steps_per_epoch=150,
    callbacks=[early_stopping_cb]
)
```

Рисунок 1.7 – Навчання моделі

Після аналізу коду для створення нейронної мережі потрібно визначити параметри конфігурації мережі для досягнення максимально можливої точності. Існує декілька параметрів, які можна змінити, а саме:

- mbnet_v2.trainable = False – зменшує загальну кількість параметрів для навчання, прискорює процес навчання.
- Batch_size – визначає кількість зображень, які будуть оброблятися одночасно в одному проході навчання.
- Epochs – визначає кількість повних проході через навчальний набір під час навчання.

Після навчання нейронної мережі потрібно зберегти модель та конвертувати її в формат .tflite відповідно до рис. 1.5

```
model.save('my_model.h5')
model = load_model('my_model.h5')

model = tf.keras.models.load_model('my_model.h5')

converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
quantized_tflite_model = converter.convert()

tflite_model = converter.convert()

with open('mnv3.tflite', 'wb') as f:
    f.write(tflite_model)
```

Рисунок 1.5 – Збереження та конвертація моделі

Порівняльний аналіз причин конвертації моделі з формату .h5 в формат .tflite відповідно до таблиці 3.

Таблиця 3 – порівняння h5 та tflite

Особливість	.h5	.tflite
Призначення	Зберігання моделей в Python	Виконання моделей на мобільних пристроях
Вміст	Структура моделі, ваги, параметри, конфігурація	Скомпільована версія моделі, готова до використання
Використання	В середовищі Python з TensorFlow або Keras	На мобільних та вбудованих пристроях з TensorFlow Lite
Оптимізація	Не оптимізований для мобільних пристроїв	Оптимізований для мобільних пристроїв з обмеженими ресурсами

Висновки до першого розділу

В розділі 1 описано роль нейронних мереж в задача комп'ютерного зору та класифікації зображень. Проведено порівняльний аналіз різних типів нейронних

мереж з яких було обрано згорткову нейронну мережу для вирішення поставленої задачі. Проведено порівняльний аналіз архітектур нейронних мереж та було обрано архітектуру MobileNetV2 через її високу точність та малі вимоги до обчислювальних можливостей системи. Був описаний процес створення нейронної мережі та вибір датасету, мови програмування та середовище розробки. Розглянуто ключові параметри конфігурації нейронної мережі для досягнення максимальної точності, а також принцип збереження моделі та подальша її конвертація в формат, який підтримується мобільними пристроями

Розділ 2: Розробка інформаційної системи для розпізнавання хвороб рослин

2.1 Функціональні характеристики інформаційної системи

Основною функцією інформаційної системи є класифікація зображень листових пластин для виявлення можливої хвороби використовуючи навчену нейронну мережу. Але система також повинна мати деякі можливості, а саме:

1. Завантаження зображення – користувач повинен мати змогу завантажити зображення з галереї або сфотографувати листову пластину в застосунку
2. Попередня обробка зображення – застосунок повинен обробляти зображення, а саме масштабувати, обрізати або нормалізувати
3. Класифікація зображення – після обробки зображення застосунок має визначити клас рослини
4. Візуалізація результатів – застосунок повинен мати зрозумілий інтерфейс для відображення результатів класифікації. Повинні візуалізовуватися клас рослини, впевненість в результаті та можливе лікування
5. Збереження результатів – застосунок повинен зберігати результати класифікації в окремому меню
6. Швидкість розпізнавання – застосунок використовуючи нейронну мережу має швидко та точно розпізнавати хвороби

Окрім вище зазначених функцій, застосунок повинен включати можливість реєстрації користувачів та Google Sign-In, як варіант входу в застосунок.

2.2 Розробка та реалізація архітектури інформаційної системи

Перед розробкою андроїд застосунку потрібно переглянути та зрозуміти архітектуру майбутнього проєкту. Враховуючи те, що нейронна мережа була навчена на локальному середовищі та була конвертована в формат .tflite для подальшої інтеграції в мобільний застосунок, тому архітектура системи має наступний вигляд:

1. Клієнтська частина
    - Інтерфейс користувача з можливістю завантаження або фотографування зображень листових пластин
    - Попередня обробка – масштабування, обрізання, нормалізація
    - Інтеграція нейронної мережі для запуску моделі на пристрої
    - Відображення результатів класифікації
2. Сховище пристрою
    - Зберігання результатів класифікації

Інтегрування нейронної мережі прямо в застосунок потрібне для того, щоб процес класифікації займав якомога менше часу та для забезпечення класифікації при відсутності інтернет з'єднання.

Використання нейронної мережі у форматі TFLite в мобільному додатку:

- Навчання нейронної мережі відбувається на локальному середовищі з використанням TensorFlow
- Після навчання відбувається конвертація моделі у формат TFLite
- Конвертована TFLite модель інтегрується прямо в мобільний додаток
- У додатку використовується TensorFlow Lite API для завантаження і запуску моделі TFLite.
- Коли користувач завантажує або фотографує зображення листової пластини, воно проходить попередню обробку прямо на пристрої.
- Оброблене зображення подається на вхід нейронної мережі TFLite для класифікації безпосередньо на пристрої

- Результат класифікації відображається в інтерфейсі

Така архітектура дозволяє виконувати вивід нейронної мережі на мобільному пристрої без використання серверної частини. Локальне середовище використовується лише для навчання моделі.

Для реалізації архітектури інформаційної системи буде використовуватися застосунок Android Studio та мова програмування Kotlin. Пере реалізацією система потрібно зробити, щоб застосунок використовував камеру пристрою, галерею та інтернет з'єднання для можливості авторизації через Google Sign-in. Для цього потрібно в AndroidManifest.xml додати параметри відповідно до рис. 2.1.

```
<uses-feature
    android:name="android.hardware.camera"
    android:required="false" />

<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Рисунок 2.1 – Дозвіл на використання

Відповідно код використовує дозволи на камеру, інтернет та запис та считування внутрішнього сховища, а також використовує функцію камери. Далі потрібно зробити, щоб запитувалися дозволи в застоснку для цього потрібно додати код відповідно до рис. 2.2

```
private fun requestPermissions() {
    val permissionsToRequest = arrayOf(
        android.Manifest.permission.CAMERA,
        android.Manifest.permission.READ_EXTERNAL_STORAGE
    )

    val REQUEST_CODE = 10
    requestPermissions(permissionsToRequest, REQUEST_CODE)
}
```

Рисунок 2.2 – Запит на дозіл

Даний код запитує у користувача дозвіл на використання камери та галереї під час використання додадку. Також потрібно створити дві функції для камери та гелереї, щоб коли користувач зробить знімок або обере його з галереї результат отримувався в метод OnActivityResult відповідно до рис. 2.3.

```
private fun captureFromCamera() {
    val cameraIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
    startActivityForResult(cameraIntent, CAMERA_REQUEST_CODE)
}

private fun pickFromGallery() {
    val galleryIntent = Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI)
    startActivityForResult(galleryIntent, GALLERY_REQUEST_CODE)
}
```

Рисунок 2.3 – Функції камери та галереї

Далі потрібно створити метод OnActivityResult, який буде обробляти результати отримані з камери або галереї відповідно до рис. 2.4.

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (resultCode == RESULT_OK && data != null) {
        when (requestCode) {
            CAMERA_REQUEST_CODE -> {
                val imageBitmap = data.extras?.get("data") as Bitmap
                val resizedBitmap = Bitmap.createScaledBitmap(imageBitmap, IMAGE_SIZE, IMAGE_SIZE, filter: true)
                imageView.setImageBitmap(resizedBitmap)
                classifyImage(resizedBitmap)
            }

            GALLERY_REQUEST_CODE -> {
                val imageUri = data.data
                val imageBitmap = MediaStore.Images.Media.getBitmap(this.contentResolver, imageUri)
                val resizedBitmap = Bitmap.createScaledBitmap(imageBitmap, IMAGE_SIZE, IMAGE_SIZE, filter: false)
                imageView.setImageBitmap(resizedBitmap)
                classifyImage(resizedBitmap)
            }
        }
    }
}
```

Рисунок 2.4 – Метод OnActivityResult

В даному методі результати отримані з камери масштабуються до задоного розміру, в нашому випадку 224x224 пікселів та виводиться у ImageView для відображення зображення в інтерфейсі користувача. Та викликається функція classifyImage. Для камери спочатку зображення отримується в форматі Uri, а потім конвертується в формат Bitmap. Далі зображення масштабується до заданих розмірів, виводиться у ImageView та викликається функція classifyImage.

2.3 Інтерфейс користувача та інтеграція з системою розпізнавання

Перед використанням моделі нейронної мережі в застосунку та інтеграції Google Sign-In потрібно додати залежності тобто бібліотеки до проєкту. Для цього потрібно в файлі build.grades.kts додати потрібний код відповідно до рис. 2.5.

```
implementation(platform("com.google.firebase:firebase-bom:32.8.0"))
implementation("com.google.firebase:firebase-ml-modeldownloader")
implementation("com.google.firebase:firebase-analytics")
implementation("com.google.firebase:firebase-auth:21.1.0")
implementation("com.google.android.gms:play-services-auth:21.0.0")
implementation("org.tensorflow:tensorflow-lite:2.3.0")
implementation("com.google.firebase:firebase-auth-ktx:22.3.1")
```

Рисунок 2.5 – Залежності в проєкті

Дані залежності потрібні для аутентифікації користувачів, збору аналітичних даних про використання застоснку, для завантаження та керування моделями машинного навчання та виконання моделі на мобільних пристроях.

Для інтерграції нейронної мережі в папку Assets потрібно перенести модель .tflite, текстовий документ з табличним списком класів рослин та можливе лікування для них відповідно до рис. 2.6.

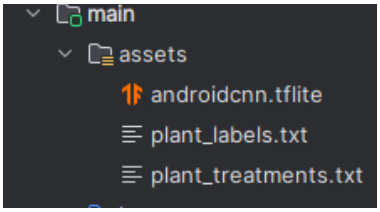

Рисунок 2.6 – Assets

Спочатку завантажимо мітки класів рослин, лікування та модель tflite відповідно до рис. 2.7.

```
labels = loadLabels( filename: "plant_labels.txt")
treatments = loadTreatments( filename: "plant_treatments.txt")

tflite = Interpreter(loadModelFile())
```

Рисунок 2.7 – Завантаження класів та моделі

Тепер потрібно завантажити файл моделі та визначити початкове зміщення та оголошену довжину файла потім виконати метод map() FileChannel для відображення вмісту файлу в пам’ять у режимі лише для читання відповідно до рис. 2.8.

```
private fun loadModelFile(): MappedByteBuffer {
    val fileDescriptor = assets.openFd( fileName: "androidcnn.tflite")
    val inputStream = FileInputStream(fileDescriptor.fileDescriptor)
    val fileChannel = inputStream.channel
    val startOffset = fileDescriptor.startOffset
    val declaredLength = fileDescriptor.declaredLength
    return fileChannel.map(FileChannel.MapMode.READ_ONLY, startOffset, declaredLength)
}
```

Рисунок 2.8 – Завантаження файлу моделі

Тепер потрібно створити дві функції для завантаження текстових даних, а саме класів та лікування відповідно до рис. 2.9.

Рисунок 2.9 – Функції тестових даних

Функція loadLabels: приймає ім'я файлу як параметр і завантажує рядки з цього текстового далі використовуючи BufferedReader, функція ітерується по рядках файлу і додає кожен рядок до списку labels.

Функція loadTreatments: має таку саму логіку, як loadLabels, але працює з іншим текстовим файлом із директорії assets. Вона завантажує рядки з файлу, ітеруючись по них за допомогою BufferedReader і додає кожен рядок до списку treatments.

Тепер потрібно створити функцію classifyImage, яка буде класифікувати зображення використовуючи модель tflite та виводити результати в інтерфейс користувача відповідно до рис. 2.10.

```
@SuppressLint("SetTextI18n")
private fun classifyImage(bitmap: Bitmap) {
    try {
        val input = Array( size: 1) { Array(IMAGE_SIZE) { Array(IMAGE_SIZE) { FloatArray( size: 3) } } }
        for (i in 0 ≤ until < IMAGE_SIZE) {
            for (j in 0 ≤ until < IMAGE_SIZE) {
                val pixelValue = bitmap.getPixel(i, j)
                input[0][i][j][0] = (pixelValue shr 16 and 0xFF) / 255.0f
                input[0][i][j][1] = (pixelValue shr 8 and 0xFF) / 255.0f
                input[0][i][j][2] = (pixelValue and 0xFF) / 255.0f
            }
        }

        val output = Array( size: 1) { FloatArray(NUM_CLASSES) }

        tflite.run(input, output)

        val maxValue = output[0].maxOrNull() ?: 0f

        if (maxValue > 0) {
            val predictedClassIndex = output[0].indexOfFirst { it == maxValue }
            val predictedClass = labels[predictedClassIndex]
            val confidence = maxValue
            val treatment = treatments[predictedClassIndex]

            val classificationResult = "Клас: $predictedClass \nЛікування: $treatment \nВпевненість: $confidence"
            recentSearchResults.addLast(classificationResult)

            // Виведення результатів
            textViewResult.text = "Клас: $predictedClass \nЛікування: $treatment \nВпевненість: $confidence"
        } else {
            textViewResult.text = "Не вдалося класифікувати зображення"
        }
    } catch (e: Exception) {
        textViewResult.text = "Помилка класифікації: ${e.message}"
    }
}
```

Рисунок 2.10 – Функція classifyImage

Дана функція приймає зображення для класифікації, підготовлює вхідні дані для моделі та виконує класифікацію шляхов викликання функції tflite.run. та знаходить клас з найвищою ймовірністю. Далі функція отримує інформацію про хворобу, лікування та впевненість класифікації та виводить відповідні дані в інтерфейс користувача.

Висновки до другого розділу

В розділі 2 були описані функціональні вимоги до інформаційної системи. Основними функціями є завантаження зображення, попередня обробка, класифікація зображення, візуалізація результатів, збереження результатів та швидка класифікація. Була розроблена архітектура інформаційної системи, яка скаладється з клієнтської частини та сховища, було визначено, що модель нейронної мережі буде інтегрована безпосередньо в застосунок. Відповідно до архітектури системи був описаний процес розробки та реалізації додатку, а саме інтеграцію нейронної мережі TFLite в додаток Android, налаштування дозволів для камери, галереї та інтернету, реалізацію функцій завантаження зображень, попередньої обробки та класифікації зображень.

Розділ 3: Оцінка ефективності та керівництво користувача

3.1 Оцінка ефективності системи розпізнавання хвороб рослин

Для оцінки ефективності системи потрібно провести декілька тестів використовуючи можливість завантаження зображення з галереї та безпосередньо фотографування в застосунку.

Для оцінки ефективності з використанням камери пристрою буде проведено 2 теста використовуючи класи Малина_Здорова та Полуниця_Здорова. Спочатку проведено аналіз полуниці відповідно до рис. 3.1.

Рисунок 3.1 – Аналіз полуниці

Відповідно до рисунку 3.1 застосунок визначив клас рослини з ймовірністю 90% та надав потрібне лікування.

Далі проведено аналіз малини згідно рис. 3.2.

Рисунок 3.2 – Аналіз малини

Відповідно рисунку 3.2 застосунок визначив клас рослини з ймовірністю лише 45%, тобто даний результат є правильним але з дещо низькою впевненістю.

Тестування з галереї нічим не відрізняється від тестування камери тільки те, що потрібні зображення мають зберігатися в сховищі пристрою. Зображення для тестування були обрані в мережі інтернет для класів Перець_Бактеріальна_пляма та Персик_ Бактеріальна_пляма відповідно до рис. 3.3

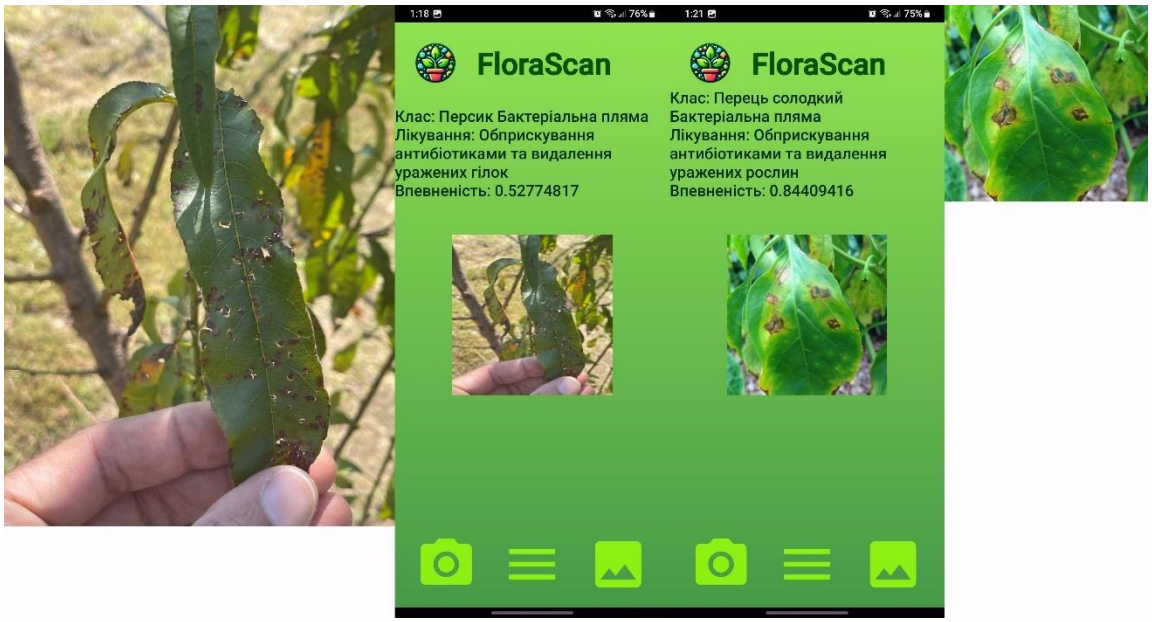

Рисунок 3.3 – Тестування

Відповідно до рисунку 3.3 застосунко визначив класи правиильно з точністю 84% та 52% відповідно.

Час, який займає виконання моделі нейронної мережі від завантаження зображення до виводу результатів користувачеві включаючи етапи попередньої обробки та класифікації, відбувається миттєво, адже правильно підібрана архітектура та отпимізований код не навантажують пристрій користувача.

3.2 Керівництво користувача

Перед написання керівництва користувача потрібно розробити діаграму послідовності для розуміння, як відбуваються процеси з погляду користувача відповідно рис. 3.4.

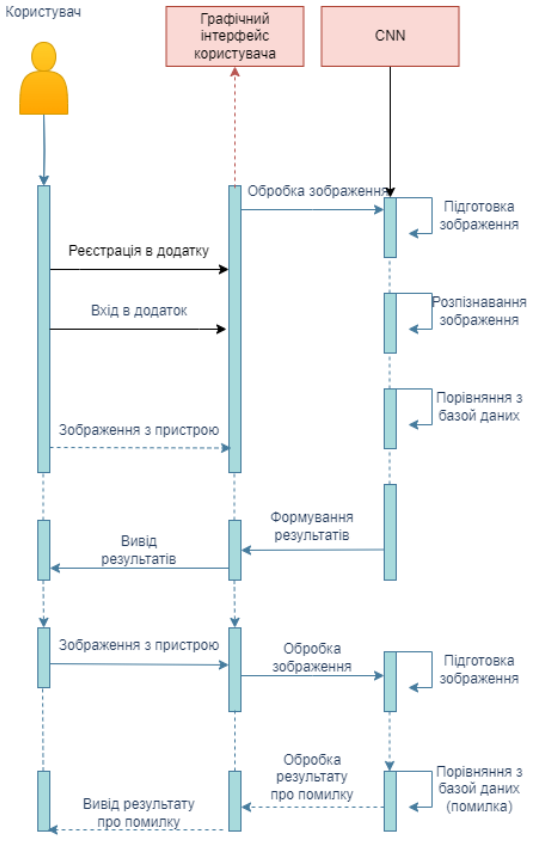

Рисунок 3.4 – Діаграма послідовності

Відповідно до діаграми користувачеві спочатку потрібно зареєструватися в додатку використовуючи логін та пароль або увійти за допомогою Google Sign-In відповідно до рис. 3.5.

Рисунок 3.5. – Аутентифікація користувача

Після аутентифікації користувач маю змогу завантажити зображення з галереї або сфотографувати безпосередньо в застосунку, також у користувача є змога переглянути попередні результати відповідно до рис. 3.6.

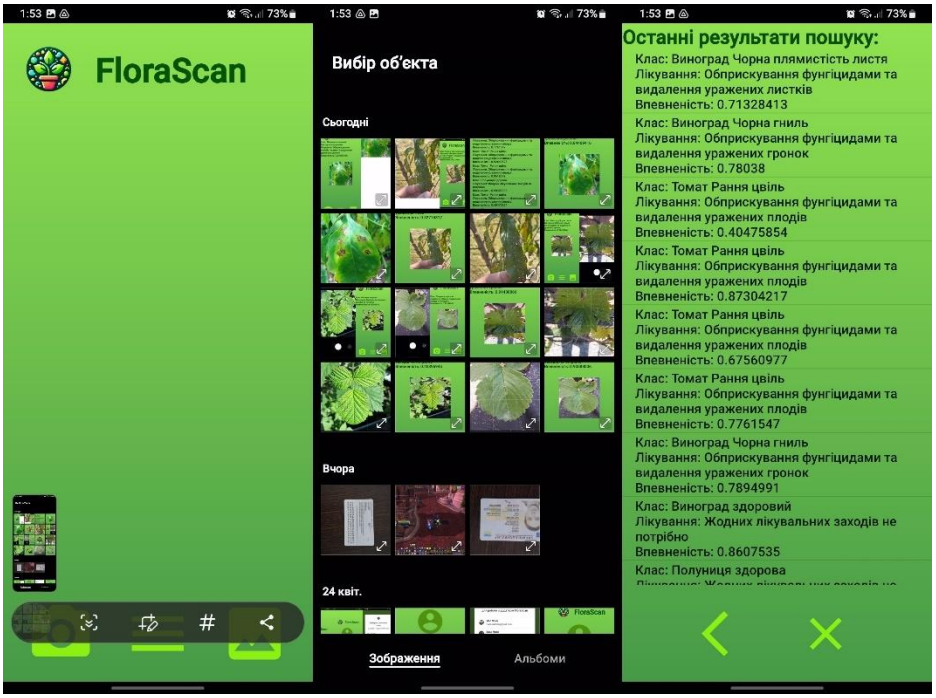

Рисунок 3.6 – Інтерфейс користувача

Відповідно до діаграми послідовності процеси класифікації та попередньої обробки та візуалізація результатів відбуваються без участі користувача, що робить додаток простим та зручним для користування.

Висновки до третього розділу

В розділі 3 система розпізнавання хвороб рослин була протестована, демонструючи змінну точності класифікації. В деяких випадках точність скаладала 90%, а в деяких була достаньо низькою – приблизно 45% але результат класифікації всеодно був правильним. Швидкість класифікації миттєва, що є перевагою для користувачів. Розроблення діаграма послідовності демонтрує взаємодію користувача з системою, де більшість етапів відбуваються не участі користувача.

Висновки

У представленій роботі було досліджено та реалізовано інформаційну систему розпізнавання хвороб рослин використовуючи зображення листової пластини з використанням нейронних мереж.

1. Обґрунтовано вибір згорткових нейронних мереж, як найбільш ефективного типу для вирішення задачі класифікації зображень. Проведено порівняльний аналіз архітектур і обрано MobileNetV2 завдяки її високій точності, ефективності використання ресурсів та низькій обчислювальній складності.

2. Детально описано процес створення нейронної мережі, включаючи вибір мови програмування, середовища розробки, налаштування параметрів навчання. Використано датасет PlantVillage для навчання моделі. Застосовано методи Data Augmentation для покращення моделі.

3. Розроблено архітектуру інформаційної системи, де нейронна мережа інтегрована безпосередньо в мобільний додаток для забезпечення швидкої класифікації без необхідності підключення до серверів.

4. Реалізовано Android додаток з інтеграцією навченої моделі у форматі TFLite. Додаток забезпечує можливість завантаження або фотографування зображень, попередньої обробки, класифікації та візуалізації результатів.

5. Проведено тестування системи на різних зображеннях. Точність класифікації коливалася від 45% до 90% в залежності від зображення. Навіть за низької впевненості модель визначала правильний клас. Час класифікації був миттєвим завдяки оптимізованій архітектурі.

6. Розроблено керівництво користувача, яке описує процес реєстрації, авторизації, завантаження зображень та візуалізацію результатів класифікації.

Загалом, створена інформаційна система є практичною реалізацією системи розпізнавання хвороб рослин на мобільних пристроях із застосуванням методів машинного навчання. Оптимізована архітектура забезпечує швидку роботу, проте існує простір для вдосконалення точності моделі шляхом збільшення обсягу та різноманітності навчальних даних.

Список використаних джерел

1. **What is machine learning and how does it work? In-depth guide**: веб-сайт. URL: https://www.techtarget.com/searchcio/definition/transfer-learning (Дата звернення 08.04.2024)
2. API Documentation: веб-сайт. URL: https://tensorflow.org/api_docs (Дата звернення 08.04.2024)
3. Install TensorFlow 2: веб-сайт. URL: https://www.tensorflow.org/install (Дата звернення 08.04.2024)
4. Keras: The high-level API for TensorFlow: веб-сайт. URL: https://www.tensorflow.org/guide/keras (Дата звернення 09.04.2024)
5. MobileNetV2 Explained: веб-сайт. URL: https://paperswithcode.com/method/mobilenetv2 (Дата звернення 08.04.2024)
6. Фітопатологія – наука про хвороби рослин та способи боротьби з ними: веб-сайт. URL: https://archer.chnu.edu.ua/bitstream/handle/123456789/6028/KONSPECT_FITOPAT_2021_11_A5.pdf?sequence=1&isAllowed=y (Дата звернення 09.04.2024)
7. What are convolutional neural networks?: веб-сайт. URL: https://www.ibm.com/topics/convolutional-neural-networks (Дата звернення 09.04.2024)
8. MobileNet, MobileNetV2, and MobileNetV3: веб-сайт. URL: https://keras.io/api/applications/mobilenet/ (Дата звернення 10.04.2024)
9. MobileNet V2: веб-сайт. URL: https://huggingface.co/docs/transformers/main/en/model_doc/mobilenet_v2 (Дата звернення 12.04.2024)
10. Using deep learning for image-based plant disease detection: веб-сайт. URL: https://doi.org/10.3389/fpls.2016.01419 (Дата звернення 13.04.2024)
11. Integrating Google Sign-In into Your Android App: веб-сайт. URL: https://developers.google.com/identity/sign-in/android/legacy-sign-in?hl=en (Дата звернення 13.04.2024)
12. Authenticate with Google on Android: веб-сайт. URL: https://firebase.google.com/docs/auth/android/google-signin?hl=en (Дата звернення 14.04.2024)
13. Quickstart for Android: веб-сайт. URL: https://www.tensorflow.org/lite/android/quickstart (Дата звернення 16.04.2024)

14. Generate model interfaces using metadata: веб-сайт. URL: https://www.tensorflow.org/lite/inference_with_metadata/codegen (Дата звернення 18.04.2024)

15. Use a custom TensorFlow Lite model on Android: веб-сайт. URL: https://firebase.google.com/docs/ml/android/use-custom-models?hl=en (Дата звернення 19.04.2024)

16. Documentation OpenFramework: веб-сайт. URL: https://openframeworks.cc/documentation/ (Дата звернення 20.04.2024)

17. Кващук Д.М., Підлужній В.В. ДІАГНОСТИКА ЗАХВОРЮВАНЬ РОСЛИН З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ РОЗПІЗНАВАННЯ ОБРАЗІВ В СИСТЕМІ ЕКОНОМІЧНОЇ БЕЗПЕКИ ФЕРМЕРСЬКИХ ДОМОГОСПОДАРСТВ: Київ, 2019. 4 с.

18. Самарський, В. В., Кузнєцов, О. Ю Використання технологій машинного навчання для діагностики хвороб рослин: Київ, 2021. 18 с.

19. Гончаренко, С. І., & Кириленко, В. В. Застосування нейронних мереж для виявлення хвороб рослин: Харків, 2020. 24 с.

20. Петрик, В. Ф., Мисловська, В. А., Сиротюк, В. В. Використання згорткових нейронних мереж для розпізнавання захворювань рослин: Хмельницький, 2020. 162 с.

21. Kotsalou, C., Tziritas, G. Integrating CNN architectures for disease classification in plant leaves.: Виго, 2021, 7 с.

22. Антонюк, А.А., Мороз, Т.А. Розробка мобільних додатків для Android за допомогою Kotlin: Львів, 2020. 101 с.

23. Новік В.С. ОБГРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБЛЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМА РОЗПІЗНАВАННЯ ХВОРОБ РОСЛИН: матеріали міжфакультетної студентської науково-практичної конференції, м. Житомир, 14 листопада 2023 р. Житомир: ПНУ, 2023. С. 2-3

24. Новік В.С. ОБГРУНТУВАННЯ ВИБОРУ АРХІТЕКТУРИ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ РОЗРОБЛЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМА РОЗПІЗНАВАННЯ ХВОРОБ РОСЛИН: матеріали всеукраїнської студентської науково-практичної конференції, м. Житомир, 10 квітня 2024 р. Житомир: ПНУ, 2024. С. 1-3