

ДОДАТОК А

Код навчання моделі класифікації

```
import json
from pathlib import Path

import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers

PROJECT_ROOT = Path(r"C:\myProjects\DiplomPlantVision")
DATASET_ROOT = PROJECT_ROOT / "dataset" / "raw" /
"train_val_test"

TRAIN_DIR = DATASET_ROOT / "train"
VAL_DIR = DATASET_ROOT / "val"
TEST_DIR = DATASET_ROOT / "test"

MODELS_DIR = PROJECT_ROOT / "models"
MODELS_DIR.mkdir(exist_ok=True)

IMG_SIZE = (224, 224)
BATCH_SIZE = 32
SEED = 42
EPOCHS = 10

def load_datasets():
    train_ds = tf.keras.utils.image_dataset_from_directory(
        TRAIN_DIR,
        image_size=IMG_SIZE,
        batch_size=BATCH_SIZE,
        shuffle=True,
        seed=SEED,
    )
```

```

val_ds = tf.keras.utils.image_dataset_from_directory(
    VAL_DIR,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=False,
)

test_ds = tf.keras.utils.image_dataset_from_directory(
    TEST_DIR,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=False,
)

class_names = train_ds.class_names

    with open(MODELS_DIR / "class_names.json", "w",
encoding="utf-8") as file:
        json.dump(class_names, file, ensure_ascii=False,
indent=2)

autotune = tf.data.AUTOTUNE

train_ds = train_ds.prefetch(buffer_size=autotune)
val_ds = val_ds.cache().prefetch(buffer_size=autotune)
                                test_ds =
test_ds.cache().prefetch(buffer_size=autotune)

return train_ds, val_ds, test_ds, class_names

def build_model(num_classes: int):
    data_augmentation = tf.keras.Sequential(
        [
            layers.RandomFlip("horizontal"),
            layers.RandomRotation(0.1),
            layers.RandomZoom(0.1),

```

```

        layers.RandomContrast(0.1),
    ],
    name="data_augmentation",
)

base_model = tf.keras.applications.MobileNetV2(
    input_shape=IMG_SIZE + (3,),
    include_top=False,
    weights="imagenet",
)

base_model.trainable = False

inputs = tf.keras.Input(shape=IMG_SIZE + (3,))
x = data_augmentation(inputs)

x =

tf.keras.applications.mobilenet_v2.preprocess_input(x)
x = base_model(x, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.25)(x)

outputs = layers.Dense(num_classes,
activation="softmax")(x)

model = tf.keras.Model(inputs, outputs)

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0
.0005),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"],
)

return model

def plot_history(history):
    plt.figure()

```

```

plt.plot(history.history["accuracy"], label="train
accuracy")

plt.plot(history.history["val_accuracy"],
label="validation accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.savefig(MODELS_DIR / "training_accuracy.png",
dpi=200)

plt.figure()
plt.plot(history.history["loss"], label="train loss")
plt.plot(history.history["val_loss"],
label="validation loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.savefig(MODELS_DIR / "training_loss.png", dpi=200)

def main():
    print("TensorFlow version:", tf.__version__)
    print("Loading datasets...")

    train_ds, val_ds, test_ds, class_names =
load_datasets()

    print("Classes:")
    for index, name in enumerate(class_names):
        print(index, name)

    model = build_model(num_classes=len(class_names))
    best_model_path = MODELS_DIR /
"plant_disease_mobilenetv2.keras"

```

```

callbacks = [
    tf.keras.callbacks.EarlyStopping(
        monitor="val_loss",
        patience=3,
        restore_best_weights=True,
    ),
    tf.keras.callbacks.ModelCheckpoint(
        filepath=best_model_path,
        monitor="val_loss",
        save_best_only=True,
        verbose=1,
    ),
]

print("Training model...")
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=EPOCHS,
    callbacks=callbacks,
)

print("Evaluating on test set...")
test_loss, test_accuracy = model.evaluate(test_ds)

print(f"Test loss: {test_loss:.4f}")
print(f"Test accuracy: {test_accuracy:.4f}")

model_path = best_model_path

metrics = {
    "test_loss": float(test_loss),
    "test_accuracy": float(test_accuracy),
    "epochs_planned": EPOCHS,
    "epochs_trained": len(history.history["loss"]),
    "image_size": IMG_SIZE,
}

```

```
        "batch_size": BATCH_SIZE,
        "num_classes": len(class_names),
    }

    with open(MODELS_DIR / "metrics.json", "w",
encoding="utf-8") as file:
        json.dump(metrics, file, ensure_ascii=False,
indent=2)

    plot_history(history)

    print("Model saved to:", model_path)
    print("Metrics saved to:", MODELS_DIR / "metrics.json")

if __name__ == "__main__":
    main()
```

ДОДАТОК Б

Код донавчання моделі класифікації

```
import json
from pathlib import Path

import matplotlib.pyplot as plt
import tensorflow as tf

PROJECT_ROOT = Path(r"C:\myProjects\DiplomPlantVision")
DATASET_ROOT = PROJECT_ROOT / "dataset" / "raw" /
"train_val_test"

TRAIN_DIR = DATASET_ROOT / "train"
VAL_DIR = DATASET_ROOT / "val"
TEST_DIR = DATASET_ROOT / "test"

MODELS_DIR = PROJECT_ROOT / "models"

BASE_MODEL_PATH = MODELS_DIR /
"plant_disease_mobilenetv2.keras"
FINE_TUNED_MODEL_PATH = MODELS_DIR /
"plant_disease_mobilenetv2_finetuned.keras"

IMG_SIZE = (224, 224)
BATCH_SIZE = 32
SEED = 42
FINE_TUNE_EPOCHS = 10
UNFREEZE_LAST_LAYERS = 35

def load_datasets():
    train_ds = tf.keras.utils.image_dataset_from_directory(
        TRAIN_DIR,
        image_size=IMG_SIZE,
        batch_size=BATCH_SIZE,
```

```

        shuffle=True,
        seed=SEED,
    )

    val_ds = tf.keras.utils.image_dataset_from_directory(
        VAL_DIR,
        image_size=IMG_SIZE,
        batch_size=BATCH_SIZE,
        shuffle=False,
    )

    test_ds = tf.keras.utils.image_dataset_from_directory(
        TEST_DIR,
        image_size=IMG_SIZE,
        batch_size=BATCH_SIZE,
        shuffle=False,
    )

    autotune = tf.data.AUTOTUNE

    train_ds = train_ds.prefetch(buffer_size=autotune)
    val_ds = val_ds.cache().prefetch(buffer_size=autotune)
    test_ds = test_ds.cache().prefetch(buffer_size=autotune)

    return train_ds, val_ds, test_ds

def find_mobilenet_layer(model):
    for layer in model.layers:
        if isinstance(layer, tf.keras.Model) and "mobilenet"
in layer.name.lower():
            return layer

    raise ValueError("MobileNetV2 base model layer was not
found.")

```

```

def plot_fine_tuning_history(history):
    plt.figure()
    plt.plot(history.history["accuracy"], label="fine-
tuning train accuracy")
    plt.plot(history.history["val_accuracy"], label="fine-
tuning validation accuracy")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.grid(True)
    plt.savefig(MODELS_DIR / "fine_tuning_accuracy.png",
dpi=200)

    plt.figure()
    plt.plot(history.history["loss"], label="fine-tuning
train loss")
    plt.plot(history.history["val_loss"], label="fine-
tuning validation loss")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.legend()
    plt.grid(True)
    plt.savefig(MODELS_DIR / "fine_tuning_loss.png",
dpi=200)

def main():
    print("TensorFlow version:", tf.__version__)
    print("Loading datasets...")

    train_ds, val_ds, test_ds = load_datasets()

    print("Loading base model...")
    model = tf.keras.models.load_model(BASE_MODEL_PATH)

    base_model = find_mobilenet_layer(model)

```

```

print("Base model layer:", base_model.name)
        print("Total layers in base model:",
len(base_model.layers))

base_model.trainable = True

for layer in base_model.layers[:-UNFREEZE_LAST_LAYERS]:
    layer.trainable = False

for layer in base_model.layers[-UNFREEZE_LAST_LAYERS:]:
    layer.trainable = True

print(f"Unfrozen last layers: {UNFREEZE_LAST_LAYERS}")

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0
.00001),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"],
)

callbacks = [
    tf.keras.callbacks.EarlyStopping(
        monitor="val_loss",
        patience=3,
        restore_best_weights=True,
    ),
    tf.keras.callbacks.ModelCheckpoint(
        filepath=FINE_TUNED_MODEL_PATH,
        monitor="val_loss",
        save_best_only=True,
        verbose=1,
    ),
]

print("Fine-tuning model...")

```

```

history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=FINE_TUNE_EPOCHS,
    callbacks=callbacks,
)

print("Evaluating fine-tuned model on test set...")
test_loss, test_accuracy = model.evaluate(test_ds)

print(f"Fine-tuned test loss: {test_loss:.4f}")
print(f"Fine-tuned test accuracy: {test_accuracy:.4f}")

metrics = {
    "base_model_path": str(BASE_MODEL_PATH),
    "fine_tuned_model_path":
str(FINE_TUNED_MODEL_PATH),
    "test_loss": float(test_loss),
    "test_accuracy": float(test_accuracy),
    "fine_tune_epochs_planned": FINE_TUNE_EPOCHS,
    "fine_tune_epochs_trained":
len(history.history["loss"]),
    "unfrozen_last_layers": UNFREEZE_LAST_LAYERS,
    "learning_rate": 0.00001,
    "image_size": IMG_SIZE,
    "batch_size": BATCH_SIZE,
}

with open(MODELS_DIR / "fine_tuned_metrics.json", "w",
encoding="utf-8") as file:
    json.dump(metrics, file, ensure_ascii=False,
indent=2)

plot_fine_tuning_history(history)

```

```
        print("Fine-tuned model saved to:",  
FINE_TUNED_MODEL_PATH)  
        print("Metrics saved to:", MODELS_DIR /  
"fine_tuned_metrics.json")  
  
if __name__ == "__main__":  
    main()
```

ДОДАТОК В

Код реалізації модуля Grad-CAM

```

import base64
import io

import cv2
import numpy as np
import tensorflow as tf
from PIL import Image

IMG_SIZE = (224, 224)

def get_model_parts(model):
    data_augmentation =
model.get_layer("data_augmentation")
    base_model = model.get_layer("mobilenetv2_1.00_224")
    pooling_layer =
model.get_layer("global_average_pooling2d")
    dropout_layer = model.get_layer("dropout")
    dense_layer = model.get_layer("dense")

    return data_augmentation, base_model, pooling_layer,
dropout_layer, dense_layer

def make_gradcam_heatmap(image_array, model,
pred_index=None):
    (
        data_augmentation,
        base_model,
        pooling_layer,
        dropout_layer,
        dense_layer,
    ) = get_model_parts(model)

```

```

last_conv_layer = base_model.get_layer("Conv_1")

inputs = tf.keras.Input(shape=IMG_SIZE + (3,))

x = data_augmentation(inputs, training=False)
                                x                                =
tf.keras.applications.mobilenet_v2.preprocess_input(x)

feature_extractor = tf.keras.Model(
    inputs=base_model.input,
    outputs=[
        last_conv_layer.output,
        base_model.output,
    ],
)

conv_outputs, x = feature_extractor(x)

x = pooling_layer(x)
x = dropout_layer(x, training=False)
predictions = dense_layer(x)

grad_model = tf.keras.Model(
    inputs=inputs,
    outputs=[conv_outputs, predictions],
)

with tf.GradientTape() as tape:
    conv_outputs, predictions = grad_model(image_array)

    if pred_index is None:
        pred_index = tf.argmax(predictions[0])

    class_channel = predictions[:, pred_index]

gradients = tape.gradient(class_channel, conv_outputs)

```

```

        pooled_gradients = tf.reduce_mean(gradients, axis=(0,
1, 2))

        conv_outputs = conv_outputs[0]

        heatmap = conv_outputs @ pooled_gradients[...,
tf.newaxis]

        heatmap = tf.squeeze(heatmap)
        heatmap = tf.maximum(heatmap, 0)

        max_value = tf.reduce_max(heatmap)

        if max_value == 0:
            heatmap = tf.zeros_like(heatmap)
        else:
            heatmap = heatmap / max_value

        return heatmap.numpy()

def create_gradcam_base64(original_image: Image.Image,
heatmap, alpha=0.4) -> str:

        original_image =
original_image.convert("RGB").resize(IMG_SIZE)
        original_array = np.array(original_image)

        heatmap = cv2.resize(heatmap, IMG_SIZE)
        heatmap = np.uint8(255 * heatmap)

        heatmap_color = cv2.applyColorMap(heatmap,
cv2.COLORMAP_JET)
        heatmap_color = cv2.cvtColor(heatmap_color,
cv2.COLOR_BGR2RGB)

        superimposed = heatmap_color * alpha + original_array
        superimposed = np.clip(superimposed, 0,
255).astype(np.uint8)

```

```
result_image = Image.fromarray(superimposed)

buffer = io.BytesIO()
result_image.save(buffer, format="PNG")

image_base64 =
base64.b64encode(buffer.getvalue()).decode("utf-8")

return f"data:image/png;base64,{image_base64}"
```

ДОДАТОК Г

Код FastAPI-сервісу інтелектуальної діагностики

```

import json
from pathlib import Path

import numpy as np
import tensorflow as tf
from fastapi import FastAPI, File, Form, HTTPException,
UploadFile
from fastapi.middleware.cors import CORSMiddleware
from PIL import Image
from app.gradcam import create_gradcam_base64,
make_gradcam_heatmap

from app.recommendations import get_recommendation

PROJECT_ROOT = Path(r"C:\myProjects\DiplomPlantVision")
MODEL_PATH = PROJECT_ROOT / "models" /
"plant_disease_mobilenetv2_finetuned.keras"
CLASS_NAMES_PATH = PROJECT_ROOT / "models" /
"class_names.json"

IMG_SIZE = (224, 224)

app = FastAPI(
    title="Plant Disease Diagnosis API",
    description="API для діагностики фітопатологій
пасльонових культур",
    version="1.0.0",
)

app.add_middleware(
    CORSMiddleware,

```

```

        allow_origins=["*"],
        allow_credentials=True,
        allow_methods=["*"],
        allow_headers=["*"],
    )

def load_class_names() -> list[str]:
    if not CLASS_NAMES_PATH.exists():
        raise FileNotFoundError(f"Class names file not
found: {CLASS_NAMES_PATH}")

    with open(CLASS_NAMES_PATH, "r", encoding="utf-8") as
file:

        return json.load(file)

def load_model():
    if not MODEL_PATH.exists():
        raise FileNotFoundError(f"Model file not found:
{MODEL_PATH}")

    return tf.keras.models.load_model(MODEL_PATH)

class_names = load_class_names()
model = load_model()

def preprocess_image(image_file) -> tuple[np.ndarray,
Image.Image]:
    try:
        image = Image.open(image_file).convert("RGB")
        resized_image = image.resize(IMG_SIZE)

        image_array = np.array(resized_image,
dtype=np.float32)
        image_array = np.expand_dims(image_array, axis=0)

    return image_array, image

```

```

except Exception as error:
    raise HTTPException(
        status_code=400,
        detail=f"Не вдалося обробити зображення:
{error}",
    )

@app.get("/health")
def health_check():
    return {
        "status": "ok",
        "model_loaded": model is not None,
        "classes_count": len(class_names),
    }

@app.post("/predict")
async def predict(
    file: UploadFile = File(...),
    temperature: float | None = Form(None),
    humidity: float | None = Form(None),
):
    if not file.content_type or not
file.content_type.startswith("image/"):
        raise HTTPException(
            status_code=400,
            detail="Файл має бути зображенням.",
        )

        image_array, original_image =
preprocess_image(file.file)
        predictions = model.predict(image_array)

        predicted_index = int(np.argmax(predictions[0]))
        confidence = float(predictions[0][predicted_index])
        class_name = class_names[predicted_index]

```

```

recommendation = get_recommendation(
    class_name,
    temperature=temperature,
    humidity=humidity,
)
heatmap = make_gradcam_heatmap(
    image_array,
    model,
    pred_index=predicted_index,
)

gradcam_image = create_gradcam_base64(
    original_image,
    heatmap,
)
return {
    "class_name": class_name,
    "culture": recommendation["culture"],
    "disease": recommendation["disease"],
    "status": recommendation["status"],
    "description": recommendation["description"],
    "confidence": confidence,
    "confidence_percent": round(confidence * 100, 2),
    "recommendation": recommendation["recommendation"],
    "gradcam_image": gradcam_image,
}

```

ДОДАТОК Д

Код модуля рекомендацій

```

RECOMMENDATIONS = {
    "Potato___Early_blight": {
        "culture": "Картопля",
        "disease": "Альтернаріоз",
        "status": "diseased",
        "description": "Виявлено ознаки ранньої плямистості картоплі.",
        "recommendation": (
            "Рекомендується видалити уражене листя, уникати надмірного "
            "зволоження ґрунту та забезпечити достатню циркуляцію повітря. "
            "За необхідності слід застосувати фунгіцидні засоби відповідно "
            "до інструкції виробника."
        ),
    },
    "Potato___Late_blight": {
        "culture": "Картопля",
        "disease": "Фітофтороз",
        "status": "diseased",
        "description": "Виявлено ознаки фітофторозу картоплі.",
        "recommendation": (
            "Рекомендується ізолювати уражені рослини, видалити пошкоджені "
            "частини та зменшити вологість у зоні вирощування. Доцільно "
            "провести обробку дозволеними фунгіцидами."
        ),
    },
    "Potato___healthy": {

```

```

    "culture": "Картопля",
    "disease": "Здоровий стан",
    "status": "healthy",
    "description": "Ознак захворювання картоплі не
виявлено.",
    "recommendation": (
        "Рекомендується продовжувати регулярний огляд
рослини, "
        "контролювати режим поливу та уникати
перезволоження."
    ),
},
"Tomato_Bacterial_spot": {
    "culture": "Томат",
    "disease": "Бактеріальна плямистість",
    "status": "diseased",
    "description": "Виявлено ознаки бактеріальної
плямистості томату.",
    "recommendation": (
        "Рекомендується видалити уражене листя, уникати
потрапляння води "
        "на листову поверхню та забезпечити
провітрювання. Варто "
        "дотримуватись санітарних заходів і не
використовувати заражений "
        "рослинний матеріал."
    ),
},
"Tomato_Early_blight": {
    "culture": "Томат",
    "disease": "Альтернатриоз",
    "status": "diseased",
    "description": "Виявлено ознаки альтернатриозу
томату.",
    "recommendation": (

```

```

        "Рекомендується видалити уражені частини
рослини, зменшити "
        "контакт листя з вологим ґрунтом і забезпечити
достатню "
        "вентиляцію. За потреби можливе застосування
фунгіцидів."
    ),
},
"Tomato_Late_blight": {
    "culture": "Томат",
    "disease": "Фітофтороз",
    "status": "diseased",
    "description": "Виявлено ознаки фітофторозу
томату.",
    "recommendation": (
        "Рекомендується швидко видалити уражені листки,
уникати "
        "перезволоження та забезпечити провітрювання.
У разі поширення "
        "симптомів слід провести фунгіцидну обробку."
    ),
},
"Tomato_Leaf_Mold": {
    "culture": "Томат",
    "disease": "Листкова пліснява",
    "status": "diseased",
    "description": "Виявлено ознаки листової плісняви
томату.",
    "recommendation": (
        "Рекомендується знизити вологість повітря,
покращити вентиляцію "
        "та уникати загущення посадок. Уражене листя
слід видалити."
    ),
},
"Tomato_Septoria_leaf_spot": {

```

```

    "culture": "Томат",
    "disease": "Септоріоз листя",
    "status": "diseased",
    "description": "Виявлено ознаки септоріозу листя
томату.",
    "recommendation": (
        "Рекомендується видалити уражене листя, уникати
поливу по листю "
        "та прибрати рослинні рештки. За потреби слід
застосувати "
        "захисну обробку."
    ),
},
"Tomato_Spider_mites_Two_spotted_spider_mite": {
    "culture": "Томат",
    "disease": "Павутинний кліщ",
    "status": "diseased",
    "description": "Виявлено ознаки ураження павутинним
кліщем.",
    "recommendation": (
        "Рекомендується оглянути нижню сторону листків,
видалити сильно "
        "уражені частини та за потреби застосувати
відповідні засоби "
        "проти кліщів."
    ),
},
"Tomato__Target_Spot": {
    "culture": "Томат",
    "disease": "Цільова плямистість",
    "status": "diseased",
    "description": "Виявлено ознаки цільової плямистості
томату.",
    "recommendation": (
        "Рекомендується видалити уражені листки,
покращити вентиляцію "

```

```

        "та контролювати вологість. За необхідності
        можливе застосування "
        "фунгіцидів."
    ),
},
"Tomato__Tomato_YellowLeaf__Curl_Virus": {
    "culture": "Томат",
    "disease": "Вірус жовтого скручування листя",
    "status": "diseased",
    "description": "Виявлено ознаки вірусу жовтого
    скручування листя томату.",
    "recommendation": (
        "Рекомендується ізолювати уражену рослину,
        контролювати комах-"
        "переносників та використовувати здоровий
        посадковий матеріал. "
        "Сильно уражені рослини доцільно видалити."
    ),
},
"Tomato__Tomato_mosaic_virus": {
    "culture": "Томат",
    "disease": "Вірус тютюнової мозаїки томату",
    "status": "diseased",
    "description": "Виявлено ознаки мозаїчного вірусного
    ураження томату.",
    "recommendation": (
        "Рекомендується видалити уражені рослини,
        дезінфікувати інструменти "
        "та уникати механічного перенесення інфекції
        між рослинами."
    ),
},
"Tomato_healthy": {
    "culture": "Томат",
    "disease": "Здоровий стан",
    "status": "healthy",

```

```

        "description": "Ознак захворювання томату не
виявлено.",
        "recommendation": (
            "Рекомендується підтримувати збалансований режим
поливу, "
            "забезпечити достатнє освітлення та регулярно
оглядати рослину."
        ),
    },
}

```

```

def add_environment_notes(
    recommendation: str,
    temperature: float | None = None,
    humidity: float | None = None,
) -> str:
    notes = []

    if humidity is not None:
        if humidity >= 80:
            notes.append(
                "За введеними параметрами вологість є
підвищеною. Такі умови "
                "можуть сприяти розвитку грибкових
захворювань, тому варто "
                "покращити вентиляцію, уникати поливу по
листя та не допускати "
                "тривалого зволоження листової поверхні."
            )
        elif humidity <= 35:
            notes.append(
                "За введеними параметрами вологість є
зниженою. Це може "
                "посилювати стрес рослини, тому
рекомендується контролювати "

```

```

        "регулярність поливу та не допускати
пересихання ґрунту."
    )

```

```

    if temperature is not None:
        if temperature >= 30:
            notes.append(
                "Температура є підвищеною, що може створювати
додатковий "
                "стрес для рослини. Доцільно уникати поливу
в найспекотніші "
                "години та забезпечити стабільний режим
зволоження."
            )
        elif temperature <= 10:
            notes.append(
                "Температура є зниженою, що може
сповільнювати ріст рослини "
                "та її відновлення після ураження. Варто
уникати різких "
                "температурних коливань і забезпечити більш
стабільні умови "
                "вирощування."
            )

```

```

    if notes:
        return recommendation + "\n\nДодатково з урахуванням
параметрів середовища: " + " ".join(notes)

```

```

    return recommendation

```

```

def get_recommendation(
    class_name: str,
    temperature: float | None = None,
    humidity: float | None = None,
) -> dict:

```

```

result = RECOMMENDATIONS.get(
    class_name,
    {
        "culture": "Невідомо",
        "disease": class_name,
        "status": "unknown",
        "description": "Опис для цього класу
відсутній.",
        "recommendation": "Рекомендується звернутися до
фахівця для уточнення стану рослини.",
    },
).copy()

result["recommendation"] = add_environment_notes(
    result["recommendation"],
    temperature=temperature,
    humidity=humidity,
)

return result

```

ДОДАТОК Е

Код вебзастосунку ASP.NET Core MVC

Е.1 – код Program.cs

```
using Microsoft.EntityFrameworkCore;
using PlantVisionWeb.Data;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlite("Data Source=plantvision.db"));

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to
    change this for production scenarios, see
    https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();
```

```

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();

```

E.2 – код HomeController.cs

```

using System.Diagnostics;
using Microsoft.AspNetCore.Mvc;
using PlantVisionWeb.Data;
using PlantVisionWeb.Models;

namespace PlantVisionWeb.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;
        private readonly ApplicationDbContext _context;

        public HomeController(
            ILogger<HomeController> logger,
            ApplicationDbContext context)
        {
            _logger = logger;
            _context = context;
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Result()
        {

```

```

        return View();
    }

    public IActionResult History()
    {
        var results = _context.DiagnosisResults
            .OrderByDescending(x => x.CreatedAt)
            .ToList();

        return View(results);
    }

    public IActionResult Privacy()
    {
        return View();
    }

    [HttpPost]
    public async Task<IActionResult>
SaveDiagnosis([FromBody] DiagnosisResult result)
    {
        if (result == null)
        {
            return BadRequest();
        }

        result.CreatedAt = DateTime.Now;

        _context.DiagnosisResults.Add(result);
        await _context.SaveChangesAsync();

        return Json(new { success = true, id = result.Id
    });
    }

```

```

        [ResponseCache(Duration = 0, Location =
ResponseCacheLocation.None, NoStore = true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel
            {
                RequestId = Activity.Current?.Id ??
HttpContext.TraceIdentifier
            });
        }
    }
}

```

E.3 – код DiagnosisResult.cs

```

using System;

namespace PlantVisionWeb.Models
{
    public class DiagnosisResult
    {
        public int Id { get; set; }

        public string Culture { get; set; } = string.Empty;

        public string Disease { get; set; } = string.Empty;

        public string Status { get; set; } = string.Empty;

        public double ConfidencePercent { get; set; }

        public string Description { get; set; } =
string.Empty;
    }
}

```

```

        public string Recommendation { get; set; } =
string.Empty;

        public string? ObservationDate { get; set; }

        public double? Temperature { get; set; }

        public double? Humidity { get; set; }

        public string ImageData { get; set; } =
string.Empty;

        public DateTime CreatedAt { get; set; } =
DateTime.Now;
    }
}

```

E.4 – код ApplicationDbContext.cs

```

using Microsoft.EntityFrameworkCore;
using PlantVisionWeb.Models;

namespace PlantVisionWeb.Data
{
    public class ApplicationDbContext : DbContext
    {
        public ApplicationDbContext(
            DbContextOptions<ApplicationDbContext> options
        ) : base(options)
        {
        }

        public DbSet<DiagnosisResult> DiagnosisResults {
get; set; }
    }
}

```

}

ДОДАТОК Ж

Код сторінок користувацького інтерфейсу

Ж.1 – код Index.cshtml

```
@{
    ViewData["Title"] = "Діагностика";
}

<div class="header">
    <div class="header-left">
        <div class="logo">  </div>
        <div>
            <div class="app-name">PlantVision</div>
            <div class="app-sub">Моніторинг фітопатологій
пасльонових культур</div>
        </div>
    </div>

    <div class="header-right">
        <a class="history-link" href="/Home/History">
            Історія перевірок
        </a>
    </div>
</div>

<div class="main-container">

    <div class="side-block">
        <h3>Вибір культури</h3>
        <p>Оберіть культуру для аналізу</p>

        <div class="plant-list">
            <label class="plant-item">
```

```

        <input      type="radio"      name="culture"
value="Tomato" />

        <span>🍅 Томат</span>
    </label>

    <label class="plant-item">
        <input      type="radio"      name="culture"
value="Potato" />

        <span>🥔 Картопля</span>
    </label>
</div>

<label                                class="form-label">Дата
спостереження</label>
    <input      class="form-input"      type="date"
id="dateInput" />

<p>Необов'язкові дані для уточнення
рекомендацій</p>

<label class="form-label">Температура, °C</label>
    <input      class="form-input"      type="number"
id="temperatureInput" placeholder="Наприклад, 24" step="1" />

    <label class="form-label">Вологість, %</label>
    <input      class="form-input"      type="number"
id="humidityInput"    placeholder="Наприклад, 70"    min="0"
max="100" step="1" />
</div>

<div class="upload-block">
    <h2>Завантаження зображення</h2>
    <p>Завантажте фото листка рослини для аналізу</p>

    <div class="upload-area" id="uploadArea">

```

```

        <input          type="file"          id="fileInput"
accept="image/*" hidden />

        <div id="uploadText">
            Натисніть для завантаження або перетягніть
файл сюди
            <br />
            <span>Підтримуються формати JPG, JPEG,
PNG</span>
        </div>

        <img id="previewImage" class="preview-image"
style="display:none;" />
    </div>

    <button class="secondary-btn" type="button"
onclick="document.getElementById('fileInput').click()">
        📁 Обрати файл
    </button>

    <div id="errorBox" class="error-box"
style="display:none;"></div>

    <button class="main-btn" type="button"
onclick="validateForm()">
        Аналізувати
    </button>

    <div class="tip">
        <strong>Порада:</strong><br />
        Для точнішої діагностики використовуйте чітке
фото листка при хорошому освітленні.
        Листок має займати більшу частину кадру.
    </div>
</div>

```

```

<div class="side-block">
  <h3>Результат аналізу</h3>
  <p id="resultHint">Тут з'явиться короткий результат
після аналізу</p>

  <div id="quickResult" class="result-placeholder">
    🌱
  </div>
</div>

</div>

<div class="footer">
  <div class="footer-header">
    <h3>Загальні рекомендації</h3>
    <p>Базові поради для підтримки здорового стану
рослин</p>
  </div>

  <div class="footer-content">
    <div class="footer-col">
      <h4>💧 Полив</h4>
      <p>Уникайте перезволоження ґрунту та
потрапляння води на листя.</p>
    </div>

    <div class="footer-col">
      <h4>☀️ Освітлення</h4>
      <p>Забезпечте достатнє освітлення та уникайте
різких змін умов вирощування.</p>
    </div>

    <div class="footer-col">
      <h4>🌱 Огляд</h4>

```

```

        <p>Регулярно перевіряйте листя на наявність
        плям, деформацій або зміни кольору.</p>

```

```

    </div>

```

```

</div>

```

```

</div>

```

```

@section Scripts {

```

```

    <script>

```

```

        let selectedFile = null;

```

```

        const fileInput =
document.getElementById("fileInput");

```

```

        const uploadArea =
document.getElementById("uploadArea");

```

```

        const previewImage =
document.getElementById("previewImage");

```

```

        const uploadText =
document.getElementById("uploadText");

```

```

        const errorBox =
document.getElementById("errorBox");

```

```

        uploadArea.addEventListener("click", function () {
            fileInput.click();
        });

```

```

        fileInput.addEventListener("change", function
(event) {
            const file = event.target.files[0];
            handleSelectedFile(file);
        });

```

```

        uploadArea.addEventListener("dragover", function
(event) {
            event.preventDefault();
            uploadArea.classList.add("upload-area-
active");

```

```

    });

    uploadArea.addEventListener("dragleave", function
() {
        uploadArea.classList.remove("upload-area-
active");
    });

    uploadArea.addEventListener("drop", function
(event) {
        event.preventDefault();
        uploadArea.classList.remove("upload-area-
active");

        const file = event.dataTransfer.files[0];
        handleSelectedFile(file);
    });

    function handleSelectedFile(file) {
        if (!file) {
            return;
        }

        if (!file.type.startsWith("image/")) {
            showError("Будь ласка, завантажте файл
зображення.");
            return;
        }

        selectedFile = file;

        const reader = new FileReader();

        reader.onload = function (e) {
            previewImage.src = e.target.result;
            previewImage.style.display = "block";

```

```

        uploadText.style.display = "none";
        hideError();
    };

    reader.readAsDataURL(file);
}

async function validateForm() {
    const culture =
document.querySelector('input[name="culture"]:checked');
    const date =
document.getElementById("dateInput").value;
    const temperature =
document.getElementById("temperatureInput").value;
    const humidity =
document.getElementById("humidityInput").value;

    if (!culture && !selectedFile) {
        showError("Будь ласка, оберіть культуру та
завантажте фото.");
        return;
    }

    if (!culture) {
        showError("Будь ласка, оберіть
культуру.");
        return;
    }

    if (!selectedFile) {
        showError("Будь ласка, завантажте фото
листка.");
        return;
    }
}

```

```

        if (temperature !== "" && (Number(temperature)
< -10 || Number(temperature) > 60)) {
            showError("Температура має бути в межах від
-10 до 60 °C.");
            return;
        }

        if (humidity !== "" && (Number(humidity) < 0 ||
Number(humidity) > 100)) {
            showError("Вологість має бути в межах від
0 до 100%.");
            return;
        }

        hideError();

        const quickResult =
document.getElementById("quickResult");
        const resultHint =
document.getElementById("resultHint");

        resultHint.style.display = "none";
        quickResult.classList.remove("result-
placeholder");
        quickResult.innerHTML = `
            <h3>Виконується аналіз...</h3>
            <p class="muted">Зачекайте, модель
обробляє зображення.</p>
        `;

        const formData = new FormData();
        formData.append("file", selectedFile);

        if (temperature !== "") {
            formData.append("temperature",
temperature);

```

```

    }

    if (humidity !== "") {
        formData.append("humidity", humidity);
    }
    try {
        const response = await
fetch("http://127.0.0.1:8000/predict", {
            method: "POST",
            body: formData
        });

        if (!response.ok) {
            throw new Error("Не вдалося отримати
результат аналізу.");
        }

        const result = await response.json();

        const selectedCultureName = culture.value
=== "Tomato" ? "Томат" : "Картопля";

        if (result.culture !==
selectedCultureName) {
            resultHint.style.display = "none";
            quickResult.classList.remove("result-
placeholder");

            quickResult.innerHTML = `
                <h3>⚠️ Невідповідність
культури</h3>
                <p>Обрана культура не відповідає
завантаженому зображенню.</p>
                <p><strong>Ви обрали:</strong>
${selectedCultureName}</p>
            `;
        }
    }
}

```

```

        <p><strong>Модель
визначила:</strong> ${result.culture}</p>
        <p class="muted">Будь ласка,
перевірте правильність вибору культури.</p>
        `;
        return;
    }
    const imageBase64 = await
fileToBase64(selectedFile);

    const diagnosisData = {
        image: imageBase64,
        gradcamImage: result.gradcam_image,
        culture: selectedCultureName,
        date: date,
        temperature: temperature,
        humidity: humidity,
        className: result.class_name,
        disease: result.disease,
        status: result.status,
        description: result.description,
        confidencePercent:
result.confidence_percent,
        recommendation: result.recommendation
    };

    localStorage.setItem("lastDiagnosisResult",
JSON.stringify(diagnosisData));

    const statusText = result.status ===
"healthy"
        ? "🟢 Здорове"
        : "🔴 Уражене захворюванням";

```

```

        quickResult.innerHTML = `
            <h3>Результат аналізу</h3>
            <p><strong>Стан:</strong>
${statusText}</p>
            <p><strong>Впевненість:</strong>
${result.confidence_percent}%</p>
            <button class="main-btn small-btn"
type="button" onclick="goToDetailedResult()">
                Детальний результат
            </button>
        `;
    }
    catch (error) {
        showError(error.message);
        quickResult.innerHTML = "🌿";
        quickResult.classList.add("result-
placeholder");
        resultHint.style.display = "block";
    }
}

function fileToBase64(file) {
    return new Promise((resolve, reject) => {
        const reader = new FileReader();

        reader.onload = function (event) {
            resolve(event.target.result);
        };

        reader.onerror = function () {
            reject(new Error("Не вдалося зчитати
зображення."));
        };

        reader.readAsDataURL(file);
    });
}

```

```

        });
    }

    function goToDetailedResult() {
        window.location.href = "/Home/Result";
    }

    function showError(message) {
        errorBox.innerText = message;
        errorBox.style.display = "block";
    }

    function hideError() {
        errorBox.innerText = "";
        errorBox.style.display = "none";
    }
</script>
}

```

Ж.2 – код Result.cshtml

```

@{
    ViewData["Title"] = "Детальный результат";
}

<div class="header">
    <div class="header-left">
        <div class="logo">  </div>
        <div>
            <div class="app-name">PlantVision</div>
            <div class="app-sub">Детальный результат
диагностики</div>
        </div>
    </div>

```

```

        <div class="header-right">
            <a          class="history-link"          href="/">Нова
перевірка</a>
        </div>
    </div>

    <div class="result-page">

        <div class="result-image-block">
            <h3>Завантажене зображення</h3>

            <img id="resultImage" class="image-preview" />

            <div id="gradcamBlock" class="gradcam-block">
                <h3>Пояснення рішення моделі</h3>
                <img id="gradcamImage" class="image-preview"
/>

                <p class="muted">
                    Теплова карта показує ділянки зображення,
які найбільше вплинули на рішення моделі.
                </p>
            </div>

            <p><strong>Культура:</strong>                <span
id="cultureResult"></span></p>
            <p id="dateRow"><strong>Дата:</strong>        <span
id="dateResult"></span></p>
            <p
id="temperatureRow"><strong>Температура:</strong>        <span
id="temperatureResult"></span></p>
            <p id="humidityRow"><strong>Вологість:</strong>
<span id="humidityResult"></span></p>
        </div>

        <div class="result-info-block">
            <h2>Результат діагностики</h2>

```

```

        <div class="disease-card" id="diagnosisCard">
            <h3 id="statusResult"></h3>
            <p><strong>Діагноз:</strong> <span
id="diseaseResult"></span></p>
            <p><strong>Рівень впевненості:</strong> <span
id="confidenceResult"></span></p>
        </div>

        <div class="treatment-block">
            <h3>Опис</h3>
            <p id="descriptionResult"></p>
        </div>

        <div class="treatment-block">
            <h3>Рекомендації</h3>
            <p id="recommendationResult"></p>
        </div>

        <div class="result-actions">
            <button class="secondary-btn" type="button"
onclick="saveResult()">
                 Зберегти результат
            </button>

            <a href="/" class="main-btn result-back-btn">
                Повернутися на головну
            </a>
        </div>
    </div>

    @section Scripts {
        <script>

```

```

const rawData =
localStorage.getItem("lastDiagnosisResult");

if (!rawData) {
    window.location.href = "/";
}

const data = JSON.parse(rawData);

document.getElementById("resultImage").src =
data.image;
if (data.gradcamImage) {
    document.getElementById("gradcamImage").src =
data.gradcamImage;
} else {

document.getElementById("gradcamBlock").style.display = "none";
}

document.getElementById("cultureResult").innerText
= data.culture || "не вказано";

setOptionalField("dateRow", "dateResult",
data.date, "");
setOptionalField("temperatureRow",
"temperatureResult", data.temperature, " °C");
setOptionalField("humidityRow", "humidityResult",
data.humidity, "%");

const statusText = data.status === "healthy"
    ? "🟢 Здорова рослина"
    : "🔴 Рослина уражена захворюванням";

const statusColor = data.status === "healthy" ?
"#2e7d32" : "#c62828";

```

```

        document.getElementById("statusResult").innerText
= statusText;

document.getElementById("statusResult").style.color           =
statusColor;

        document.getElementById("diseaseResult").innerText
= data.disease;

document.getElementById("confidenceResult").innerText         =
data.confidencePercent + "%";

document.getElementById("descriptionResult").innerText        =
data.description;

document.getElementById("recommendationResult").innerText     =
data.recommendation;

        function setOptionalField(rowId, valueId, value,
suffix) {
            const row = document.getElementById(rowId);
            const          valueElement          =
document.getElementById(valueId);

            if (!value) {
                row.style.display = "none";
                return;
            }

            valueElement.innerText = value + suffix;
        }

        async function saveResult() {
            try {

```

```

const response = await
fetch("/Home/SaveDiagnosis", {
    method: "POST",
    headers: {
        "Content-Type": "application/json"
    },
    body: JSON.stringify({
        culture: data.culture,
        disease: data.disease,
        status: data.status,
        confidencePercent:
Number(data.confidencePercent),
        description: data.description,
        recommendation:
data.recommendation,
        observationDate: data.date ||
null,
        temperature: data.temperature ?
Number(data.temperature) : null,
        humidity: data.humidity ?
Number(data.humidity) : null,
        imageData: data.image
    })
});

if (!response.ok) {
    throw new Error("Не вдалося зберегти
результат.");
}

const saved = await response.json();

alert("Результат успішно збережено. Номер
запису: " + saved.id);
}
catch (error) {

```

```

        alert(error.message);
    }
}    </script>
}

```

Ж.3 – код History.cshtml

```

@model List<PlantVisionWeb.Models.DiagnosisResult>

@{
    ViewData["Title"] = "Історія перевірок";
}

<div class="header">
    <div class="header-left">
        <div class="logo">  </div>
        <div>
            <div class="app-name">PlantVision</div>
            <div class="app-sub">Історія діагностичних
перевірок</div>
        </div>
    </div>

    <div class="header-right">
        <a class="history-link" href="/">На головну</a>
    </div>
</div>

<div class="history-container">

    <h2>Збережені результати</h2>

    @if (!Model.Any())
    {
        <div class="empty-history">

```

```

        Історія перевірок порожня.
    </div>
}
else
{
    @foreach (var item in Model)
    {
        <div class="history-card">

            <div class="history-image">
                
            </div>

            <div class="history-info">

                <h3>@item.Disease</h3>

                <p>
                    <strong>Культура:</strong>
                    @item.Culture
                </p>

                <p>
                    <strong>Стан:</strong>
                    @(item.Status == "healthy"
                        ? "Здорова
рослина"
                        : "Уражена
рослина")
                </p>

                <p>
                    <strong>Впевненість:</strong>
                    @item.ConfidencePercent%
                </p>
            </div>
        </div>
    }
}

```

```

        @if
(!string.IsNullOrEmpty(item.ObservationDate))
        {
            <p>
                <strong>Дата
спостереження:</strong>
                @item.ObservationDate
            </p>
        }

    </div>

</div>
}
}

</div>

```

ДОДАТОК И

Відомість кваліфікаційної роботи

Позначення					Найменування		Дод. відомості	
					Текстові документи			
1.					Пояснювальна записка		128 с.	