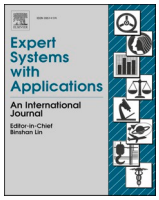
Expert Systems With Applications 289 (2025) 128304

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa



Diagnosis of orange tree fruit and leaf diseases based on a new deep learning model using a graphical user interface

Arman Foroughi [1], Jose M. Jimenez [2], Jaime Lloret [*]

Instituto de Investigacion para la Gestion Integrada de Zonas Costeras, Universitat Politecnica de Valencia, Spain

ARTICLE INFO

Keywords:
Deep learning
Kivy
Buildozer
Raspberry Pi
Orange Fruit Diseases

ABSTRACT

Identifying the spectrum of fruit and leaf disease is one of the important pillars for farmers to correctly diagnose orange disease and quickly treat it so that it does not spread to other orange trees. For this purpose, we designed an Android and iOS application that can detect orange fruit and leaf diseases, such as melanosis, black spots, canker, and greening, as well as detect healthy oranges. It also declares everything but oranges as Not Orange'. In this case, farmers no longer need specialists to diagnose fruit disease. A new deep learning model has been proposed and built to diagnose the type of orange fruit disease. KIVY framework and KV language were used for graphical programming in Android, iOS, Windows, Linux, and Raspberry Pi operating systems. To learn the proposed model, an image dataset of orange disease and orange tree leaves, containing 5073 images, was used. We have provided data images to the proposed deep learning model algorithm so that the new model can be trained by processing these images. Powerful hardware is required for image processing operations in deep learning networks. For this purpose, we used Google Collaboratory. The output of this newly trained model can be used in Windows and Linux. However, they cannot be used in mobile hardware or Android OS. Therefore, we converted the proposed new trained model to TensorFlow lite, which can be implemented in mobile phones and Android and iOS operating systems and has been optimized for this purpose. We then created an Android package using the python-for-Android project. The Buildozer tool was used to automate the entire process. The trained model could detect canker, melanosis, greening, and black spots with 98.29% accuracy. This application is easily available to farmers, who can easily detect diseases in oranges and orange tree leaves.

1. Introduction

Orange diseases caused by viral and fungal agents, pests, and bacteria cause the loss of products. It is important to develop an optimal system that can detect the type of orange fruit disease at low cost and is error-free. (Faisal et al., 2023) proposes a system for citrus disease detection using deep learning and transfer learning. In this study, pre-trained models such as EfficientNetB3, ResNet50, MobileNetV2, and InceptionV3 were trained on a citrus disease image dataset. The results showed that the EfficientNetB3 model performed the best, achieving accuracies of 99.43 % in training, 99.48 % in validation, and 99.58 % in testing. This study demonstrates that using pre-trained models can improve the accuracy of citrus disease detection and reduce computational time and resources. Additionally, this method can assist farmers in early disease detection and help prevent economic losses caused by reduced crop yields. Farmers can prevent the further spread of the disease (USDA APHIS, 2024). This reduces the loss of farmers' crops. Research in the field of artificial intelligence continues to diagnose different types of orange fruit diseases. (Nasra & Gupta, 2024) proposed a deep learning-based model using MobileNetV2 for detecting diseases in orange trees. The objective of this study is to develop an accurate and efficient method for diagnosing orange tree diseases, particularly by employing lightweight models that can be executed on mobile devices. MobileNetV2 was used as the base neural network and was trained using images of orange tree leaves. This approach can help farmers accurately identify diseases using smartphones and take timely actions for prevention and treatment.

In this study, a mobile application was developed to diagnose diseases of orange fruits and leaves. It also detects healthy fruit. In addition, the program recognizes photos other than orange fruit photos to avoid

* Corresponding author.
E-mail addresses: aforoug@doctor.upv.es (A. Foroughi), jojiher@dcom.upv.es (J.M. Jimenez), jlloret@dcom.upv.es (J. Lloret).
[1] 0009-0009-4934-5927.
[2] 0000-0002-3688-7235.

https://doi.org/10.1016/j.eswa.2025.128304
Received 12 October 2024; Received in revised form 11 May 2025; Accepted 22 May 2025
Available online 24 May 2025

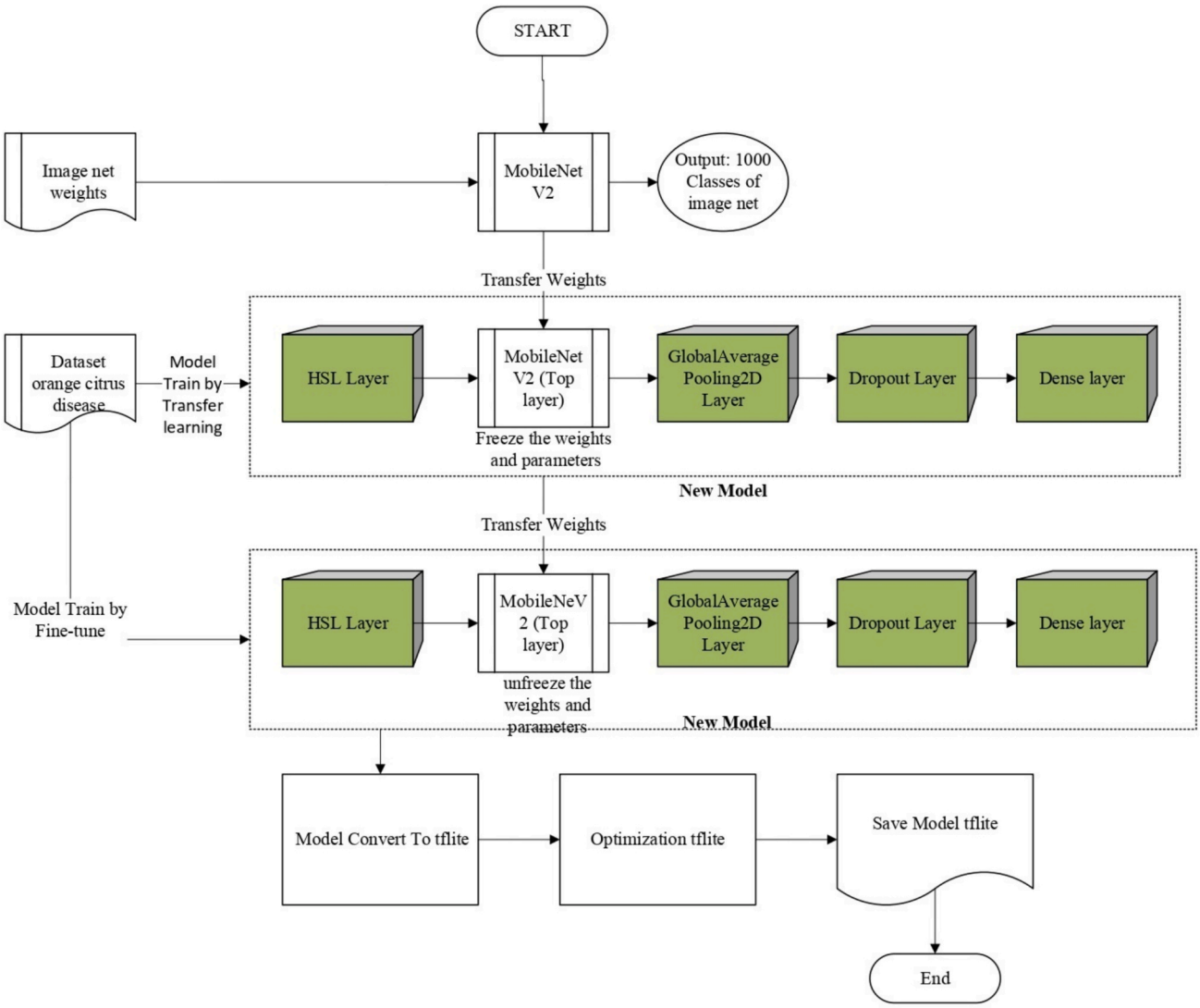


Fig. 1. New deep Learning model construction flowchart.

farmer error. It declares that this photograph is not orange. This application makes the work very easy for farmers and easily available to them. In addition, the cost of diagnosing the type of orange fruit and leaf disease for farmers was significantly reduced. In previous studies, image-processing algorithms have been used to diagnose different types of orange fruit diseases (Foroughi, Jimenez, & Lloret, 2023). However, in this study, a deep learning model was proposed to diagnose all types of orange fruit diseases with very high accuracy and minimal error. We built a new deep-learning model. We also used the TensorFlow library for the deep learning. The TensorFlow library was developed using Google (TensorFlow, n.d.). Because of the pre-trained models, it was not possible to diagnose the type of orange fruit disease (Manoharan, Thomas, & Anto Sahaya Dhas, 2021). The pre-trained deep network model that we trained for the purposes of this study using transfer learning and fine-tuning methods can diagnose orange fruit diseases in an advanced manner. To build the proposed model, we added the HSL color model to the MobileNetV2 model layers. We used the ImageNet dataset with pretrained weights (ImageNet, n.d.). Then, we froze the weights of the base model (Kumar, 2021). We loaded a network that did not include the above classification layers (Sharath et al., 2020). We built a new model based on the basic model and added the HSL color model to its lower layers, which led to a better and more advanced diagnosis of orange fruit and leaf diseases. Our model was trained with a dataset of orange fruit and leaf diseases, which included 5073 images, and the upper and lower layer weights were trained. The accuracy of the process in the first training stage was 89.15 %. After the newly constructed model converges with the new data, we unfreeze the entire base model and retrain it at a low learning rate (Malathy et al., 2021). Thus, the accuracy of orange fruit disease diagnosis increased to 98.29 %. The proposed new model was saved for use in building a graphical application. However, the proposed model cannot be implemented on Android or iOS mobile device hardware. To solve this problem, we

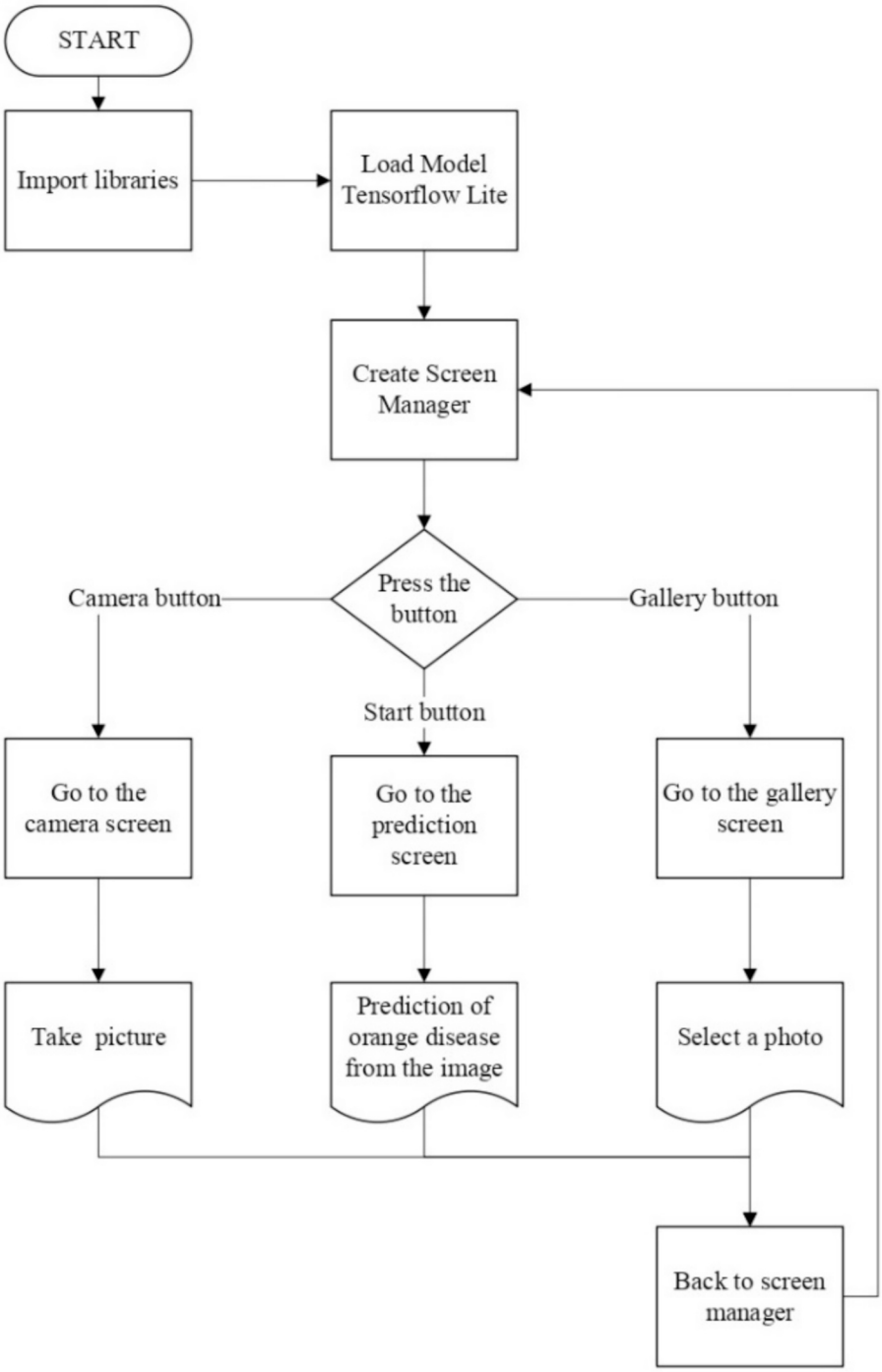


Fig. 2. Graphic User Application Flowchart for Android.

converted the proposed model into TensorFlow lite. We then used the KIVY framework in Python to design the appearance of the proposed program. It is a cross-platform, and the code is executable for Android, iOS, Windows, Linux, and Raspberry Pi devices (Saranya et al., 2020). To convert the framework codes, we used the Python-for-Android library and BUILDOZER tool (Buildozer, n.d.). Considering that if the farmers want to check the orange orchard with cameras, a separate Python program was created for the new model and the proposed application, which, by connecting to the Internet, after taking a photo of the orchard, an email containing the health and type of disease of the

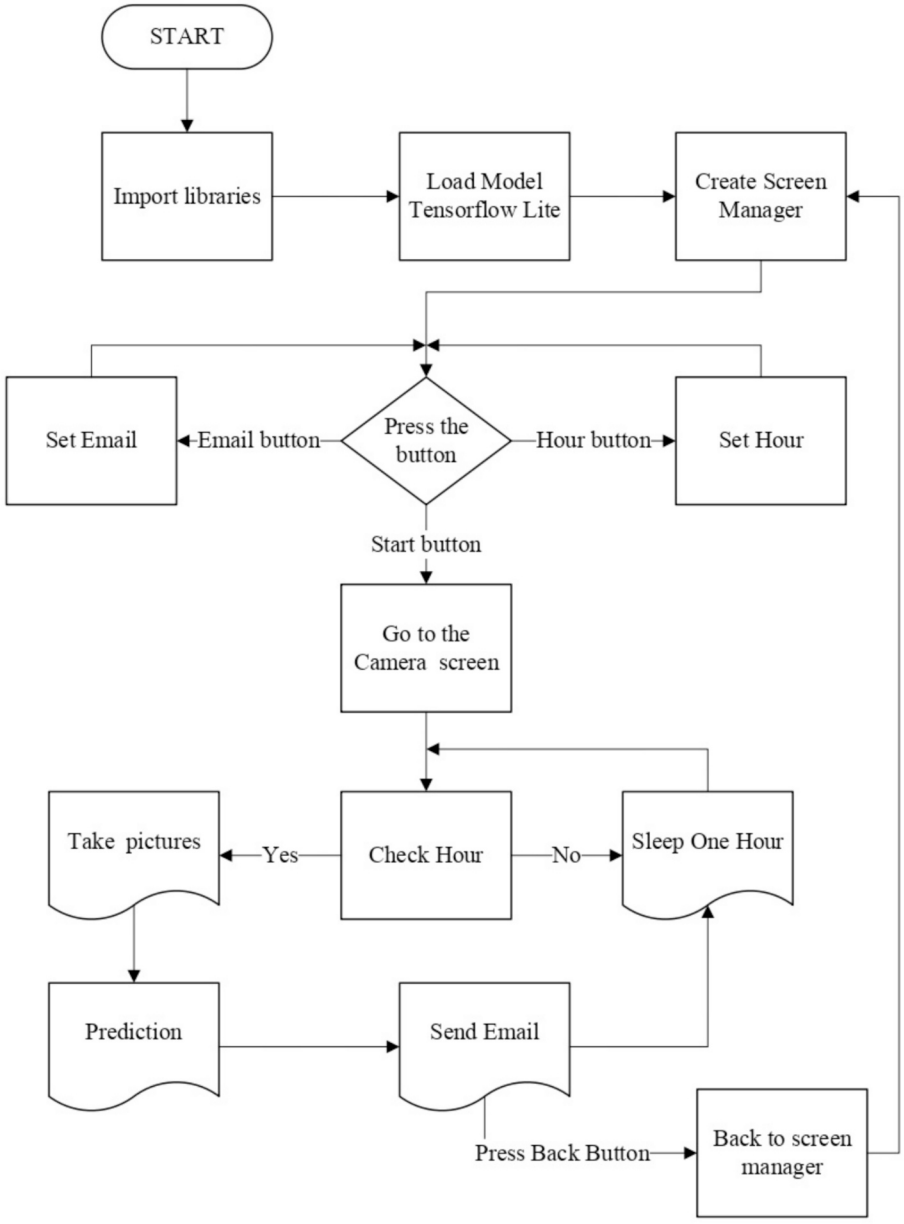


Fig. 3. Graphical user program flowchart for the Raspberry Pi and PC.

fruit and leaves Sends the orange tree to the user. The graphic program was created with the KIVY framework, where the farmer could set his email and the start time of the photography, which was implemented and tested on a Raspberry Pi 4B model.

The remainder of this article is organized as follows. In the second section, we describe the most interesting published proposals regarding deep learning algorithms for the diagnosis of plant and fruit diseases and their use in mobile applications. The third section includes building a new deep learning model and the graphic program created with the Kiwi framework, building another proposed separate program to run on Raspberry Pi and converting the proposed program to an Android program, and then explaining the flowchart and algorithm. The new program is written in Python. In the fourth section, we present the results of the new model and the diagnosis of the disease type of orange tree fruit and leaves. In the fifth section, we compare the proposed application with several other suitable applications. The results of the transfer learning and fine-tuning of the proposed deep learning model are explained in the corresponding tables. We also analyzed the processing speed of the proposed deep-learning model. We explain the advantages of the new model and the proposed application compared to those of previous studies. The proposed model was compared with four other models: Resnet50, NASNetMobile, Densenet121, and

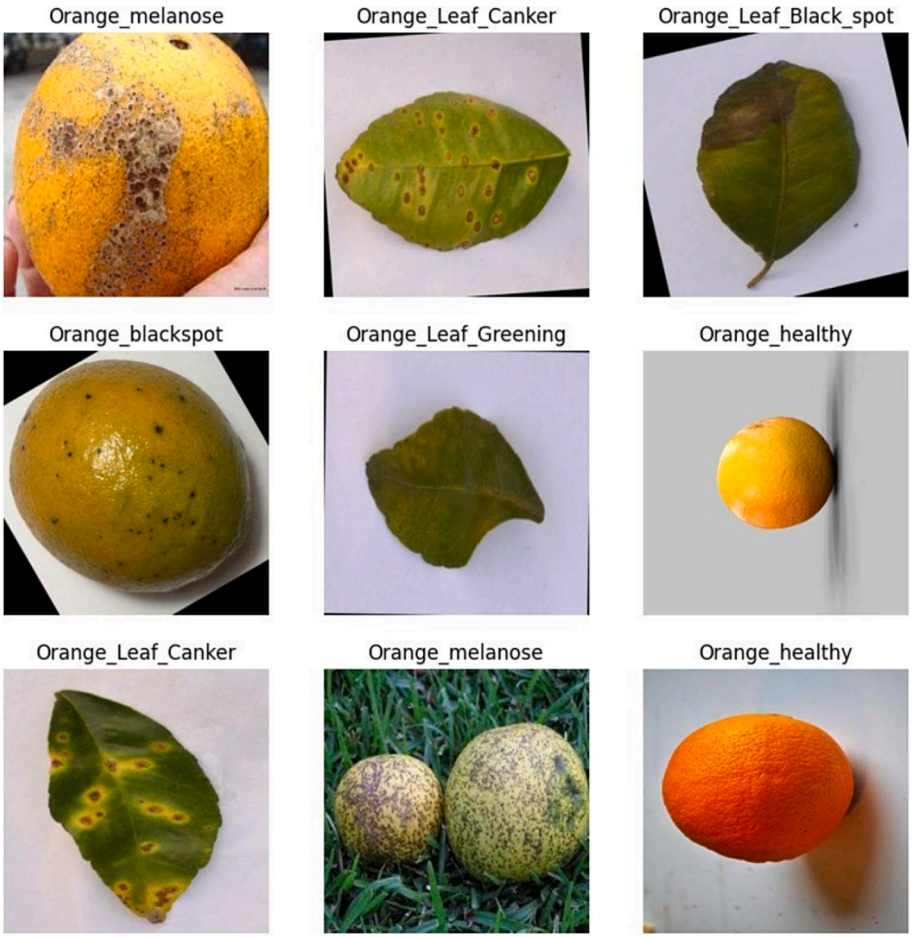


Fig. 4. Checking the labeling of photos.

MobilenetV2. We examined the results of the loss, precision, size, and total parameters. Finally, conclusions and future work are presented in the sixth section.

2. Related work

Several studies related to the creation of mobile applications for diagnosing fruit and plant leaf diseases based on CNN have been conducted. In the agricultural industry, the diagnosis of fruit or plant leaf diseases is very important to prevent farmers from losing crops. Deep learning is a suitable method for the diagnosis of fruit and plant diseases. Using a mobile application to easily diagnose fruit and leaf diseases is practical and inexpensive, and it reduces the farmers' loss of crops.

(Barman et al., 2020) compared two convolutional neural network architectures, MobileNet and self-structured (SSCNN), which were able to detect citrus leaf disease with 98 % accuracy. Their proposed system shows that SSCNN is a more accurate and useful method and requires fewer calculations. In this study, 1787 training images were used. (Nag et al., 2023) proposed a system based on a mobile application for disease detection in tomato leaves using CNN. (Nag, Chanda, & Nandi, 2023) proposed a system based on a mobile application for disease detection in tomato leaves using CNN. This study employs a transfer learning approach to fine-tune CNN architectures such as AlexNet, ResNet-50, SqueezeNet-1.1, VGG19, and DenseNet-121. All the models in this study achieved more than 95 % accuracy, with the highest accuracy for the DenseNet-121 model reaching 99.85 %. This model was proposed as a mobile application for disease detection in tomato leaves. In this study, a mobile application was developed using the Flutter framework. This program performs diagnosis in 10 s and requires an Internet connection. (Asani et al., 2023) presented a web-based plant detection program. The Kaggle dataset is used in this study. The architecture of the model had six convolutional layers. This study performed plant disease diagnosis with an accuracy of 93.91 %. In this study, a web application developed as a plant detection application with mobile capabilities was presented. This system requires the Internet to upload images and connect to the server, which is why the detection speed is low. This system could detect 15 plant diseases. (Karar et al., 2021) have proposed a new mobile application for automatic classification of pests using deep learning. The developed program is based on the faster region convolutional neural network (Faster R CNN). Five pest groups were identified. The accuracy of this study is 99.0 %. This study was done by TensorFlow framework. Also, Apache and Flask web framework software tools and MySQL and PythonAnywhere database management system have been used and a web server has been created. For this reason, their proposed application

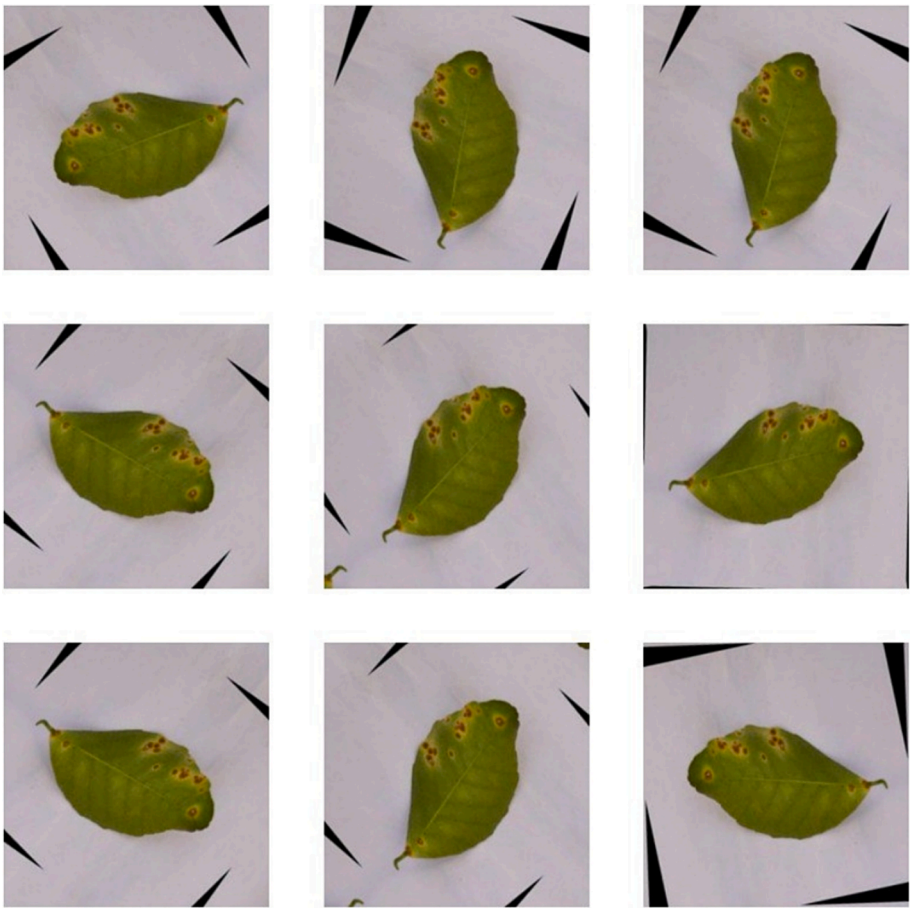


Fig. 5. Example of an augmented data image.

```
=================================================
Total params: 2257984 (8.61 MB)
Trainable params: 2223872 (8.48 MB)
Non-trainable params: 34112 (133.25 KB)
```

Fig. 6. Basic model structure of MobileNetV2.

requires internet and the detection speed is low due to uploading images and connecting to the server. (Pan et al., 2019) proposed a mobile application based on a CNN to diagnose six types of citrus diseases, which were used from DenseNet and ResNet trained networks. The accuracy of this study for the diagnosis of citrus diseases was 88 %. This study used the WeChat Applet to upload the images. They provided a CNN detection program on the Nginx Web server. The proposed system uses a dataset to train a cloud server. In this system, users upload pictures of citrus diseases to their mobile devices. This system uploads images through the WeChat applet. Then, it sends the diagnosis results and treatment recommendations to the users. This system has medium accuracy and low response speed owing to the uploading of images, and requires the use of the Internet. (Tembhurne et al., 2023) proposed a model based on MobileNet architecture along with convolutional hidden layers with keras tuner on a dataset containing 12,318 images. The proposed model classifies 64 plant disease classes for 22 different datasets of crops with an accuracy of 95.94 %. In this study, the above proposed application model has been trained in an online cloud platform Microsoft Azure. In this study, various other models were tested. For example, for InceptionResNetV2, InceptionV3, ResNet50, Xception, VGG-19, and NASNetMobile, MobileNet is the highest at 94.07 %. The proposed application is provided with Android 4.1.3 and the Flutter framework. This application uses a web server for detection, which is why it requires the Internet, and it takes time to respond. (Gencturk et al., 2023) proposed an algorithm for hazelnut classification. In this study, hazelnut classification was performed using InceptionV3 and ResNet50 models. The accuracy of hazelnut classification in this study was 100 %. The implementation of deep learning in this study was performed using MATLAB. The hazelnut studied in this research were transferred to a computer system using a camera in a bright environment. (Lanjewar & Panchbhai, 2022) proposed a system for predicting tea leaf diseases based on convolutional neural network. This cloud platform (SERVICE (PaaS) system) was the focus. For the training and validation of tea leaf prediction, the accuracy was 100 %. Among the deep neural network models ResNet and ResNet50, Xception and NASNetMobile were used in this study. In this study, the image of a tea leaf was captured with a phone camera and uploaded to the cloud System.

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 312, 312, 3)]	0
sequential (Sequential)	(None, 312, 312, 3)	0
tf.math.truediv (TFOpLambda)	(None, 312, 312, 3)	0
tf.math.subtract (TFOpLambda)	(None, 312, 312, 3)	0
mobilenetv2_1.00_224 (Functional)	(None, 10, 10, 1280)	2257984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 10)	12810

Total params: 2270794 (8.66 MB)
Trainable params: 12810 (50.04 KB)
Non-trainable params: 2257984 (8.61 MB)

Fig. 7. Model structure created using frozen parameters.

The cloud system automatically predicts tea leaf diseases and displays them on a mobile screen. In addition, this system requires the internet to upload images. Therefore, its response is not real-time. In this study, the total number of images was 120, which was very small for deep network training. (Soyer et al., 2021) proposed an android-based application to diagnose plant leaf diseases. Image processing and convolutional neural networks were used in this study. Model training was performed on a dataset of different leaf images, including 87,867 images, with an accuracy of 98.88. TensorFlow and the Keras library were used to design the model. In addition, in this study, the CNN architecture is 4 layers, each of which includes MaxPooling2D, ReLU activation, and batch normalization. In this study, CNN-based software was developed to detect plant leaf disease and integrated with an Android application. The captured image was sent to the server through the Internet and processed using the image classification model. The leaf disease information is then sent to the user. This model requires the internet to upload the image. Android user interface using programming language Java was created using Android Studio. The web server of this system is written using the Flask framework and the Python programming language. (Herman et al., 2021) proposed deep learning with convolutional neural network to detect the level of ripeness of oil palm fruits. They used the total data of seven ripening classes of oil palm fruits with 400 images. In this study, AlexNet and DenseNet learning models were used; DenseNet accuracy was 85 %, and AlexNet was presented with 80 % accuracy. (Pardede et al., 2021) studied the classification of fruit maturity. In previous studies, the maximum accuracy of the image processing technique using the HSV color model feature descriptor was only 76 %. Subsequently, they proposed a transfer learning technique using the VGG16 model. The proposed architecture used VGG16 without a top layer. Subsequently, they added a top layer to the VGG16 model. The accuracy of the proposed model increased to 95 %.

By comparing these studies, it was found that our proposed application works with a very high accuracy of 98.73 percent and without the need for a server or the Internet in android. We built a new and optimal model by combining the HSL color model and MobileNetV2 model. The accuracy of the built model was compared to of that several other models. Our model has a higher accuracy. In addition, the image loading time and diagnosis of orange fruit disease in the proposed Android application were 0.13 s. In addition, the file size of the new model of the proposed application was 2.5 MB, which is very small. This makes the disease detection process easy and inexpensive for the farmers. In addition, this application is easy for farmers to use and can be used offline. The proposed application is designed according to the needs of users on Android and iOS, and runs on Windows and Linux operating systems.

In addition, a separate program was developed for use with the Raspberry Pi device. By connecting to the Internet and taking a picture of the orange garden, it sends an email message to the user containing information about the health or type of disease fruit and leaves of the orange tree. Therefore, farmers who want to use a camera or a Raspberry Pi in the garden can use this graphic program.

3. Proposed flowchart and algorithm for diagnosing the disease type of orange tree fruits and leaves

a) Flocharts.

The flowchart of the proposed algorithm related to deep learning and model building for the diagnosis of orange tree fruit and leaf diseases is shown in Fig. 1. First, the required libraries are imported. Then we connected it to Google Drive. Then we uploaded training and validation images from Google Drive to the program. Then we labeled the images

```
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 312, 312, 3)]     0

sequential (Sequential)      (None, 312, 312, 3)       0

tf.math.truediv (TFOpLambd   (None, 312, 312, 3)       0
a)

tf.math.subtract (TFOpLamb   (None, 312, 312, 3)       0
da)

mobilenetv2_1.00_224 (Func   (None, 10, 10, 1280)      2257984
tional)

global_average_pooling2d (   (None, 1280)              0
GlobalAveragePooling2D)

dropout (Dropout)            (None, 1280)              0

dense (Dense)                (None, 10)                12810

=================================================================
Total params: 2270794 (8.66 MB)
Trainable params: 2236682 (8.53 MB)
Non-trainable params: 34112 (133.25 KB)
_________________________________________________________________
```

Fig. 8. Model structure created using unfrozen parameters.

based on the image folder. After that, we created a base model of MobileNetV2 with ImageNet weights without upper layer. Then we set the learnable parameters of the base model to False. Using this process, we freeze the weights of the base model. Then we created a top layer for our model. After that, we added a bottom layer that converts RGB to HSL. We added layers to the base model. Then we Compiled the built model. We trained the model using the entire data obtained from images of orange tree fruit and leaf diseases. In this process, only the top layer was trained. To begin fine-tuning the exerciseable weights, we unfroze the base model. We then compiled the model with a lower learning rate and trained the model again. The trained model showed a very high accuracy of 98.29 %. Then, we converted the model to tflite for use in edge and mobile hardware. Then, we optimized the tflite model and saved it for use in mobile applications. The flow chart of the kivy program is shown in Fig. 2. First, the required libraries are imported. The Tensorfelow Lite model was then loaded. The application then enters a graphical while true loop. A screen manager, which was the main page of the application, was created. The main screen has three buttons: the camera button, gallery button, and start button. It entered the camera screen by pressing the camera button. that we can capture pictures. After capturing the image, the program enters the main page again. By pressing the gallery button, you will enter the gallery screen, where you can select an image from the computer or mobile phone memory. The application then returns to the main page. By pressing the start button, we entered the orange fruit disease diagnosis page. Subsequently, the type of orange fruit disease was diagnosed. It then prints the detection type and displays the image. Subsequently, by pressing the back button, it returns to the main page. Working with this program is easy for the farmers.

It is possible that farmers will want to use CCTV cameras or Raspberry Pi cameras in large orchards to detect diseases of orange tree fruits and leaves. We designed a special program for Raspberry Pi and a PC. that sends an email message to the user after taking a picture of the fruit and leaves of the orange tree. The Raspberry Pi must be connected to the Internet. The flow diagram of the Kivy program for the Raspberry Pi and PC is shown in Fig. 3. First, libraries were imported. Subsequently, the tensor-flow lite model was loaded. The program then enters a graphical and while true loop. A screen manager, which was the main page of the program, was created. The main screen has three buttons: the email-setting, clock-setting, and start buttons. Then, under the clock setting button, we set the photography start time, and under the email setting button, the user's email is written, which is sent to this email after the diagnosis of the disease of the fruit and leaves of the orange tree. Then, by pressing the start button, it entered the camera screen. The clock is checked. If the current time is not the same as the time set by the user, the application clock will sleep for one hour and then check the desired time again. If the current time is the same as the set time, the Raspberry Pi camera or computer camera will start taking pictures, and the camera will take five pictures. Subsequently, the identification process is performed using the newly trained model. and sends the results to the user email. Then, the program went to sleep for one hour. There is also a button under the camera screen that returns the program to the screen

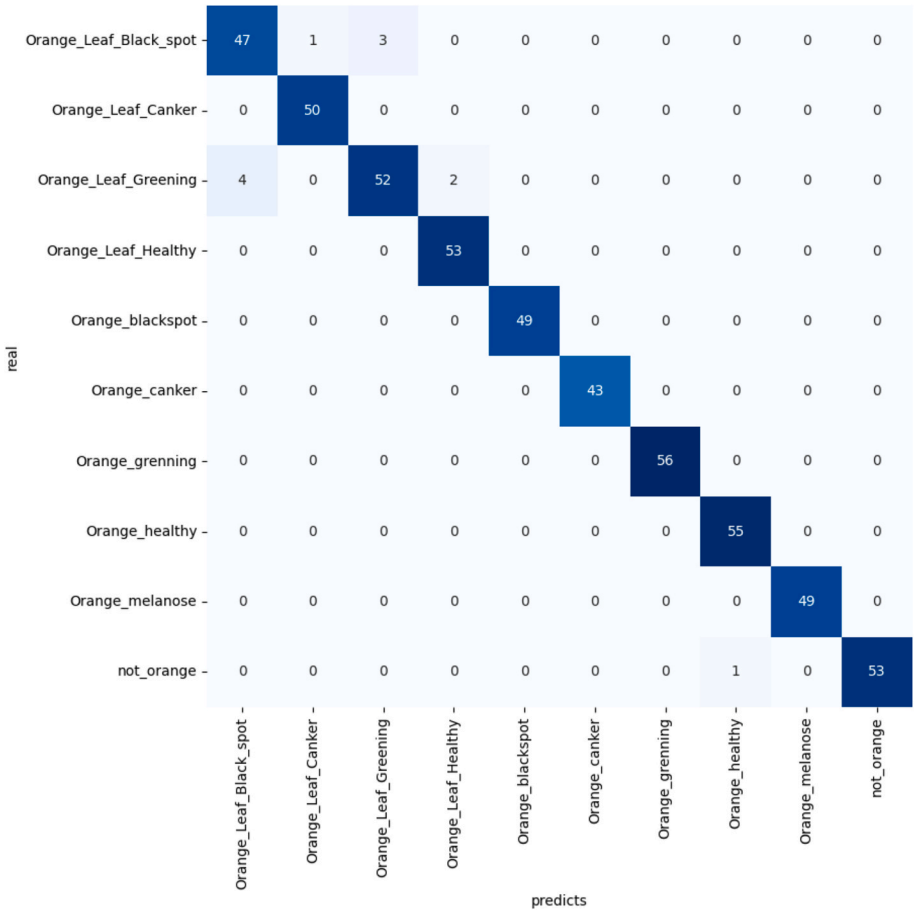


Fig. 9. Review and comparison of prediction labels and original labels.

manager screen.

b) New model deep learning algorithm.

In this section, we present a deep learning algorithm to detect fruit and leaf disease of orange trees. This is shown in the algorithm 1.

Algorithm 1

Import time, serial, os, tensorflow, keras, image, pathlib, numpy, joblib, matplotlib
Import modules **from** Google Colab **for** mounting drive
Mount Google Drive **to** access files
Define the custom layer class **RGB**to**HSL**:
Define the **RGB** to **HSL** conversion function Implement the call method **to** apply the conversion
Define the paths **for** training **and** validation directories
Set batch size **and** image size
Create training **and** validation datasets **using** image_dataset_**from**_directory()
Define class names **and** display sample images **from** the training dataset
Preprocess datasets **and** enable prefetching **for** performance optimization
Define data augmentation techniques **and** apply them **to** the training dataset
Load MobileNetV2 model without top layers **and** freeze its weights
Build a custom classification model:
Input layer,Apply RGB **to** HSL conversion, Apply data augmentation, Preprocess input for MobileNetV2, Pass through MobileNetV2 base model, Global average pooling, Dropout layer, Dense layer with softmax activation
Compile the model **with** Adam optimizer, categorical crossentropy loss, **and** accuracy metric
Train the model **for** initial epochs **and** plot training/validation accuracy **and** loss
Fine-tune the pretrained model **by** unfreezing some layers **and** adjusting learning rate
Compile and **train** the model **for** additional epochs **and** plot the updated training/validation accuracy **and** loss
Save the model structure **and** weights **to** Google Drive
Convert the model **to** TensorFlow Lite format **and** save it **to** Google Drive
Optimize the TensorFlow Lite model **for** quantization **and** save it **to** Google Drive

We require strong hardware to train the deep learning model. We used Google Colaboratory for this study (Al-Tuwaijari, Jasim, & Raheem, 2020). We then uploaded our dataset to Google Drive (Moon et al., 2021). This dataset contains two folders, train and test, each of which is placed in a separate folder. For higher processing speeds, we changed the settings of the Google Colaboratory from CPU to GPU. We wrote a code to connect Google Colaboratory to Google Drive to read the image dataset. First, we referred to necessary libraries (Foong et al., 2021). The pathlib import path library for addressing, numpy as np library for working with numbers and representations, joblib library for storing and loading files, keras preprocessing import image library for loading images, keras library for creating deep network models, mathplotlib.pyplot as plt library for drawing diagrams, os library for the operating system, and tensorflow as tf library have been used for making neural networks. Then, we saved the address of the train and test folder, which is related to the images, inside Google Drive in a variable. We then write the desired size to resize the input images in the variable (Asif et al., 2020). Then, the image_dataset_from_directory function loads the images from Google drive, categorizes them through foldering, and creates the label file (Batool et al., 2020). We then entered the class_names code, which stores a list of the names of orange fruit diseases and orange fruit health. We then wrote a for loop and displayed the fruit images with their labels using the mathploblib library (Sophia et al., 2021). We checked the labeling, as shown in Fig. 4. We created a layer for data augmentation called dataaugmentation. This process randomly creates changes, such as the rotation of images, increases the number of images, and reduces overfitting. We then wrote a For loop and displayed the data-augmentation image with mathplotlib, as shown in Fig. 5. Then we specified the shape of the images. Subsequently, we created the basic model using MobileNetV2 with imagenet weight, including_top and

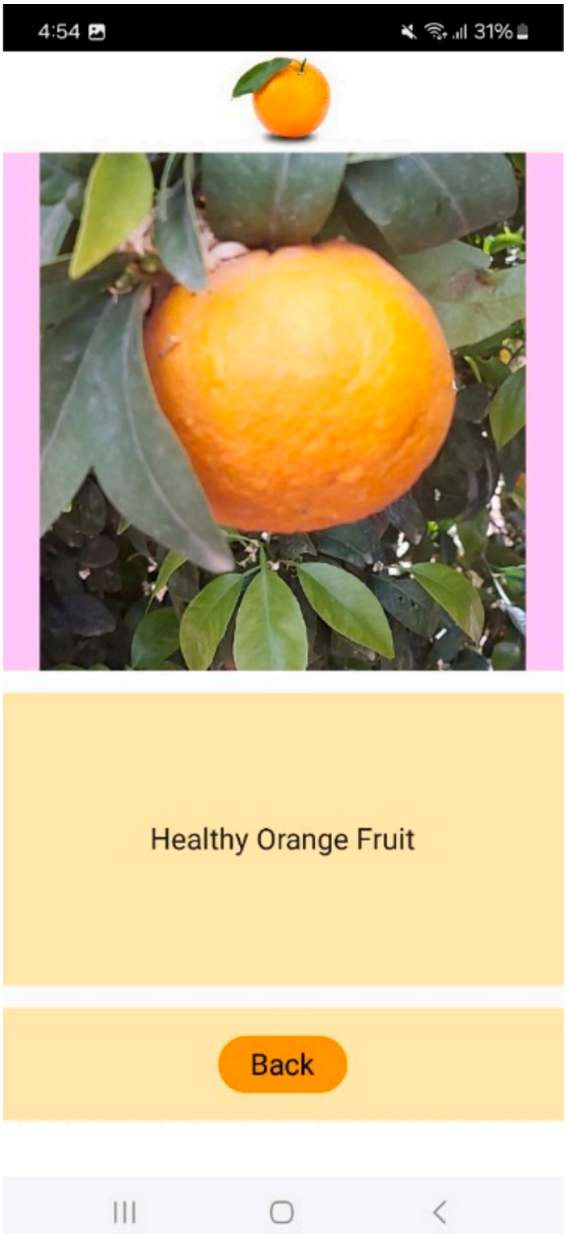


Fig. 10. Application test on orange fruit.

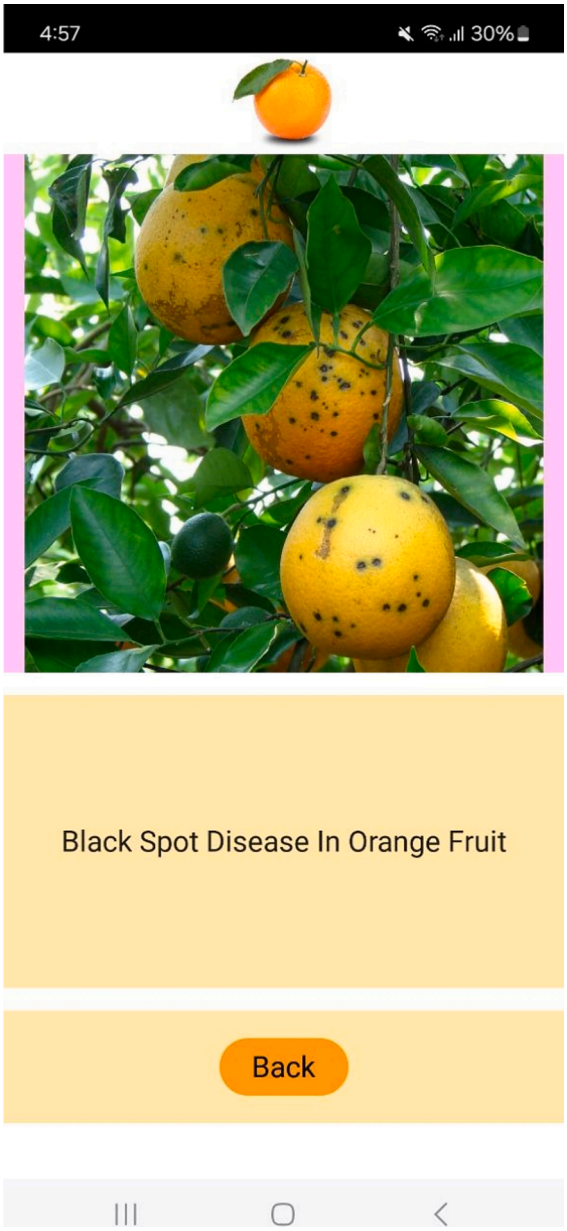


Fig. 11. Application test for orange fruit diseases.

Img_Shape = False include_top makes the top layer not be added because we want the output of the model to be set for the proposed application. We then set the trainable layers of the base model to be false (Sudana et al., 2020). This implies that the trainable weights are frozen. We wrote the base_model.summary model command, which displays the structure of the proposed model and shows the trainable, non-trainable, and total parameters, as shown in Fig. 6. Subsequently, we created an object using an input class. We aimed to create a new model. Therefore, we created a new layer by using the HSL color model. The color features of HSL make its diagnosis more accurate and optimal. In the following section, we reach this conclusion by comparing the other models. We used a data augmentation function to increase the number of images. We normalized the input data using the preprocess-input function. Subsequently, we provided inputs to our basic model and set its training to False values (Aftab et al., 2022). This means that the data pass through the basic model only once. We then added the 2D Global-AveragePoolling layer to all these data. To avoid overfitting, we added a dropout layer to all the layers. We added a dense layer with six outputs that indicated the six classes in the proposed application (Pardede et al., 2021). The final model was created using the input and output values. We then determined the learning rate and compiled the model using the Adam optimization function and the specified learning rate and loss function, categorical and metrics, accuracy. The summary model command displays the structure of the model. The number of teachable parameters was 12810, and the number of non-teachable parameters was 2257984. The total number of parameters was 2270794, as shown in Fig. 7. Then, we trained the model using training, validation, and epoch 30 data. Here, epoch specifies the number of trained loops. This model was trained with 89.15 % accuracy. Using the mathplotlib library, the graphs of accuracy and val_accuracy, loss, and val_loss were drawn. We used fine-tuning to increase accuracy. In other words, we unfreeze the frozen tunable layers and train the model at a very low learning rate. We then unfreeze all teachable layers of the base model. The number of teachable parameters was 2236682, and the number of non-teachable parameters was 34112. The total number of parameters was 2270794, as shown in Fig. 8. We then compile the model again to apply these changes. we ran the model using the fit function and dataset

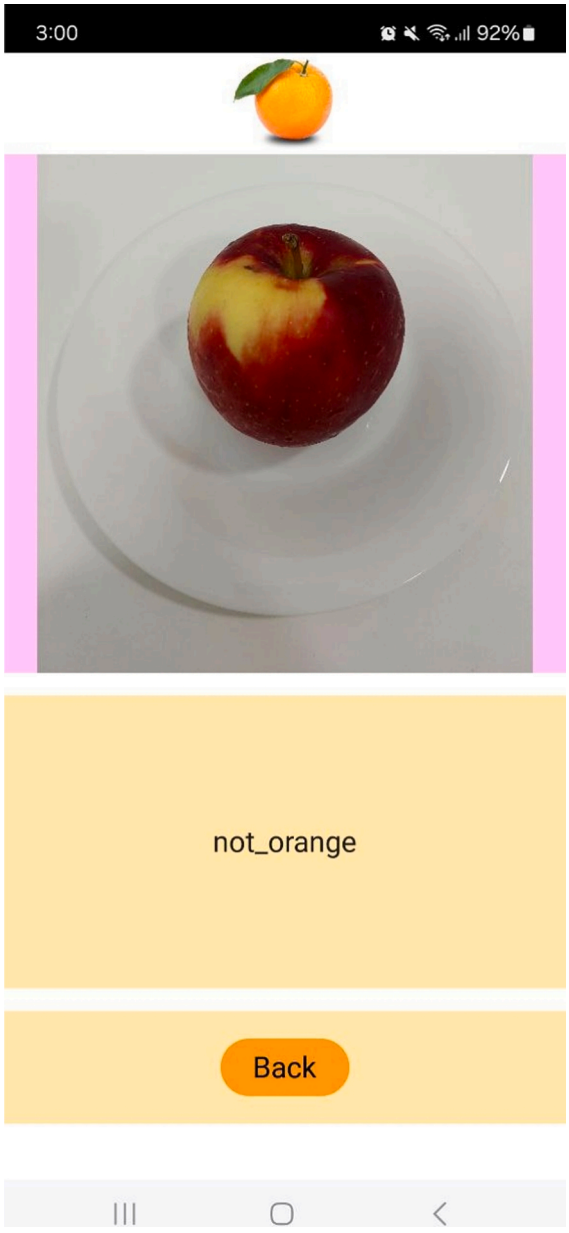


Fig. 12. Application test on apple.

parameters for training (Saragih & Emanuel, 2021). We evaluated the model, checked the accuracy and error of training and validation with matplotlib, and drew a graph. The accuracy of the model was 98.29 %. Then, we saved the model. Because the model stored inside Android devices and smartphones is not supported, we must convert the model to TensorFlow lite. Then, it should be with an extension tflite be saved because we want to reduce the size of the file and increase the speed of execution, which improves the performance of mobile phones. We optimized this file. The size of the file was reduced from 8.8 MB to 2.5 MB. We then loaded our transformed model with the TensorFelow lite interpreter. To test this, we provided a photo to the commentator for recognition. and correctly diagnosed the Canker model.

c) Mathematical formulas for newly constructed model.

We created a new model from the combination of the HSL color model and MobileNetV2 model. The HSL color model, with its color features, makes the image clearer and more optimized, and better recognition is achieved. The mathematical model of the combination of models was examined as follows.

The HSL color model is used to represent colors and consists of three components: Hue, Saturation, and Lightness. The conversion formulas from RGB to HSL are as follows.

First, we determine the maximum and minimum values among R, G, and B (1).

Max is the largest value among R, G, and B. Min is the smallest value among R, G, and B.

$$\begin{cases} Max = max(R, G, B) \\ Min = min(R, G, B) \end{cases} \tag{1}$$

Δ (Delta) is the difference between the maximum and minimum values, indicating the amount of color variation (2).

$$\Delta = Max - Min \tag{2}$$

The Lightness (L) is defined as the average of the maximum and minimum values (3). Saturation (S) indicates the purity of the color and is calculated as shown in the following formulas (4). If Delta is zero, the image is grayscale.

$$L = \frac{Max + Min}{2} \tag{3}$$

$$S = \begin{cases} if \Delta = 00 \\ if \Delta \neq 0 \dfrac{\Delta}{1 - |2L - 1|} \end{cases} \tag{4}$$

Hue (H) is the angle that determines the color on the color wheel (5).

$$H = \begin{cases} if\ \Delta = 0 & 0 \\ if\ R = Max & \left(\dfrac{G - B}{\Delta}\ \ Mod\ 6\right) \times 60 \\ if\ G = Max & \left(\dfrac{B - R}{\Delta} + 2\right) \times 60 \\ if\ B = Max & \left(\dfrac{G - R}{\Delta} + 4\right) \times 60 \end{cases} \tag{5}$$

After converting RGB to HSL, the resulting values are used as input to a neural network. The neural network used is MobileNetV2, which has 53 layers. The mathematical formulas related to the neural network are as follows (6).

$$HSL_{input} = [H(RGB), S(RGB), L(RGB)] \tag{6}$$

First Layer is Calculation of Z. The input HSL values are first combined with the weights of the first layer, W_1, and an activation function σ is applied to them to obtain Z (7).

$$Z = \sigma(HSL^{*}W_1) \tag{7}$$

Second Layer is Calculation of Y. The outputs from the first layer are combined with the weights of the second layer, W_2, and an activation function is applied to them (8).

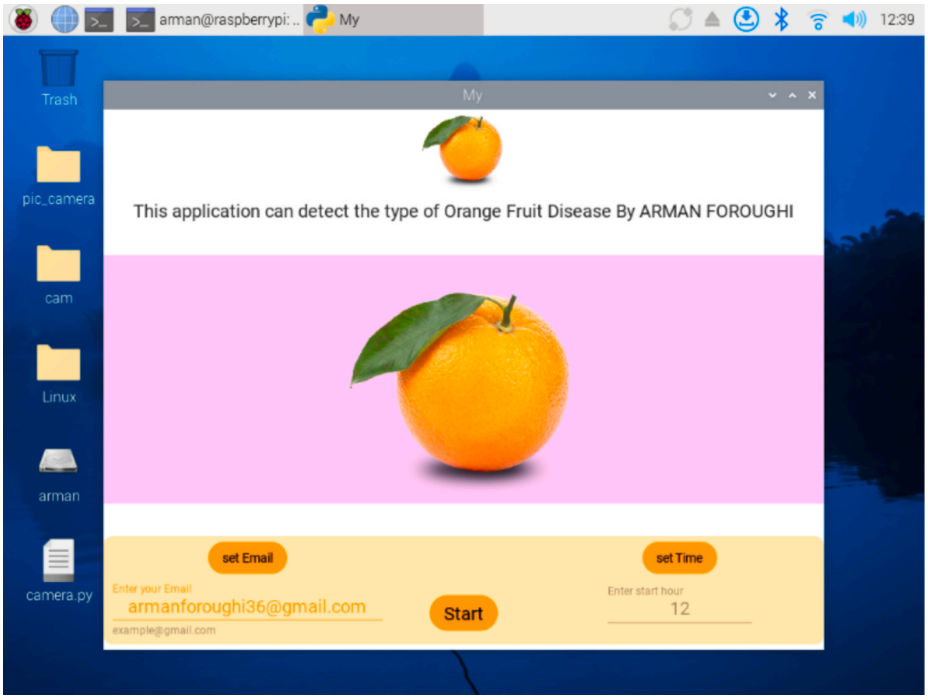


Fig. 13. Home page of the program for Raspberry Pi and PC.

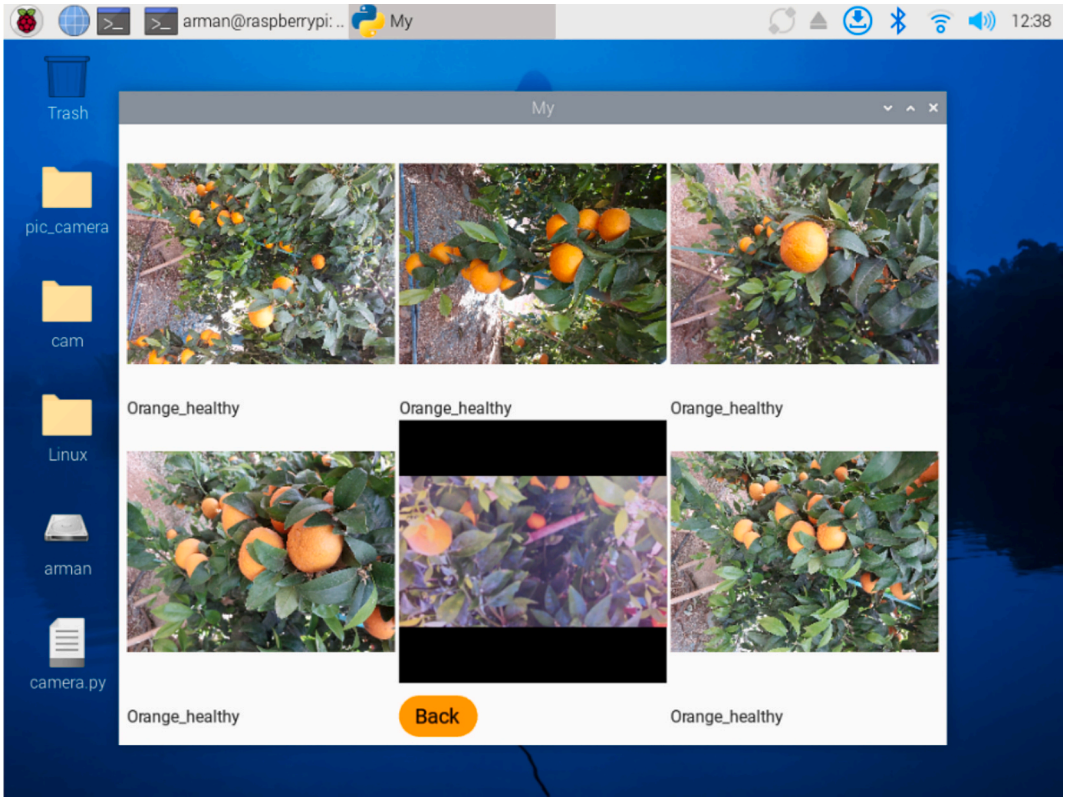


Fig. 14. The form of application in Linux environment and to detect the health of fruit and leaves of orange trees in the garden of Valencia City.

Table 1
Comparison with other studies.

Year	Author	Problem Description	Algorithm	Diagnosis type of Fruit and Plant disease	Work in offline mode	No need for a server	Accuracy
2024	Arman Foroughi	Diagnosis of orange tree fruit and leaf diseases based on a new model of Deep learning using a graphical user interface for all operating systems.	New model of CNN	Diagnosis of 4 types of orange tree fruit and leaf diseases, health and not orange (10 classes)	OK	OK	98.29 %
2020	Utpal Barman	Comparison of convolution neural networks for smartphone image based real time classification of citrus leaf disease	CNN	Citrus leaf disease	None	None	98 %
2023	Amitava Nag	Mobile app-based tomato disease identification with fine-tuned convolutional neural networks	CNN	Diagnosis of disease in tomato leaves	None	None	99.85 %
2023	Emmanuel Oluwatobi Asani	a mobile-enabled plant diseases diagnosis application using convolutional neural network toward the attainment of a food secure world	CNN	Plant disease diagnosis	None	None	93.91 %
2021	Mohamed Esmail Karar	A new mobile application of agricultural pests recognition using deep learning in cloud computing system	CNN	Automatic classification of pests	None	None	99.0 %
2019	Wenyan Pan	A Smart Mobile Diagnosis System for Citrus Diseases Based on Densely Connected Convolutional Networks	CNN	Citrus disease diagnosis	None	None	88 %
2023	Jitendra V. Tembhurne	Plant disease detection using deep learning based Mobile application	CNN	Plant disease diagnosis	None	None	95.94 %

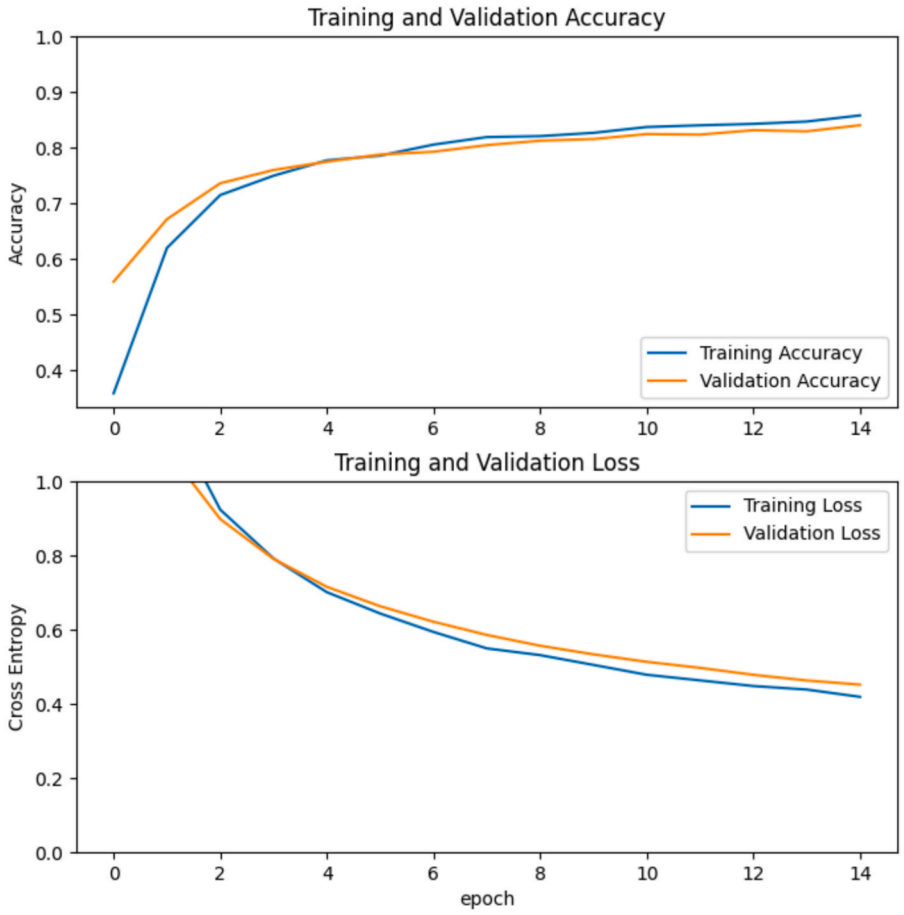


Fig. 15. Transfer Learning diagram.

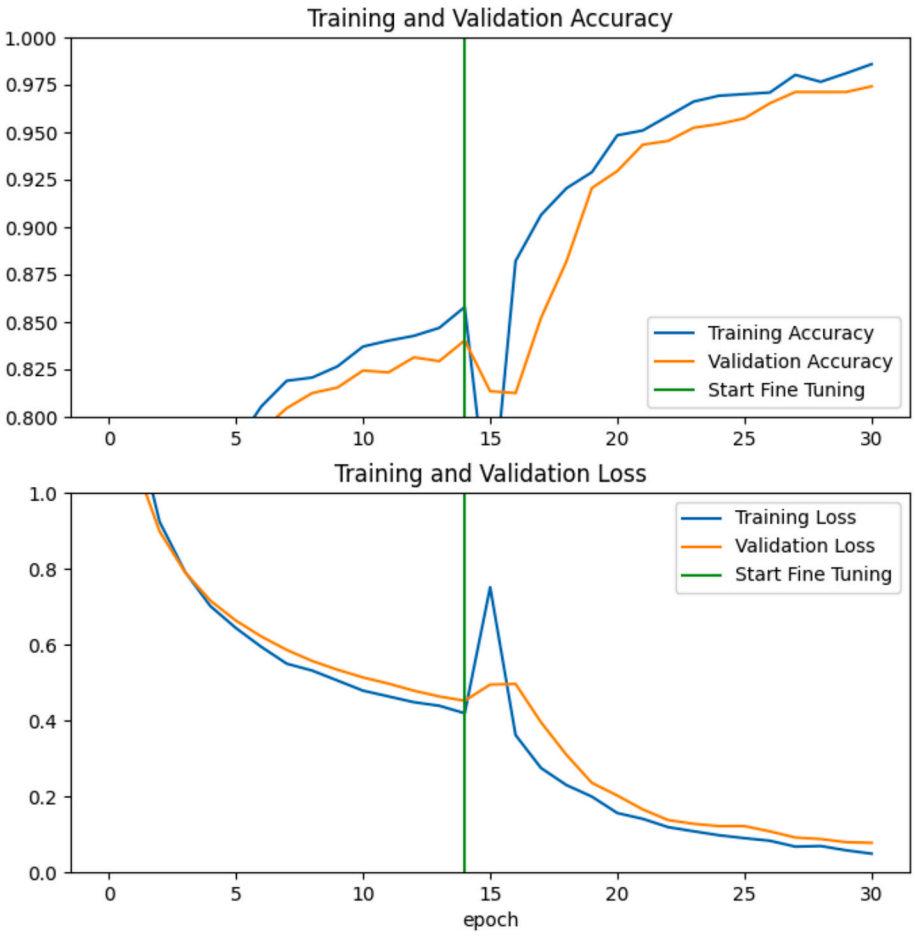


Fig. 16. Transfer Learning and Fine-tuning diagrams.

$$y = \sigma(Z_d{*}W_2) \tag{8}$$

Third Layer is Calculation of Final Outputs. The resulting values from the second layer are combined with the weights W_3 (9).

$$\acute{y} = y{*}W_3 \tag{9}$$

Final Output:

The final output values obtained from the neural network are combined with the original HSL input to produce the final output, OUT. The final output includes enhanced features that have been shown to be more optimal in comparison to several detection models.

d) Kv language for android.

The proposed application is applicable for Android and has a graphical environment. We used kivy framework and kv language. The process is presented in Algorithm 2. The codes of the proposed application design with kv language in mainkv.kv file are as follows.

Algorithm 2

Define the Manager **class with** screens MenuScreen, CameraScreen, **and** StartScreen
<MenuScreen>:
Define the name **as** "menuScreen"
Add a layout **with** a background color **and** children widgets:
– **Display** application description text
– **Display** an image
– Button **to** refresh the image
– Button **to** turn on the camera
– Button **to** pick an image **from** the gallery
– Button **to** start analyzing the selected image
<CameraScreen>:
Define the name **as** "cameraScreen"
Add a layout **with** a vertical orientation **and** a background color
Add a preview widget **for** the camera
Add buttons **for** capturing images **and** returning **to** the menu screen
<StartScreen>:
Define the name **as** "startScreen"
Add a layout **with** children widgets:
– **Display** the selected image
– **Display** the analysis result
– Button **to** return **to** the menu screen

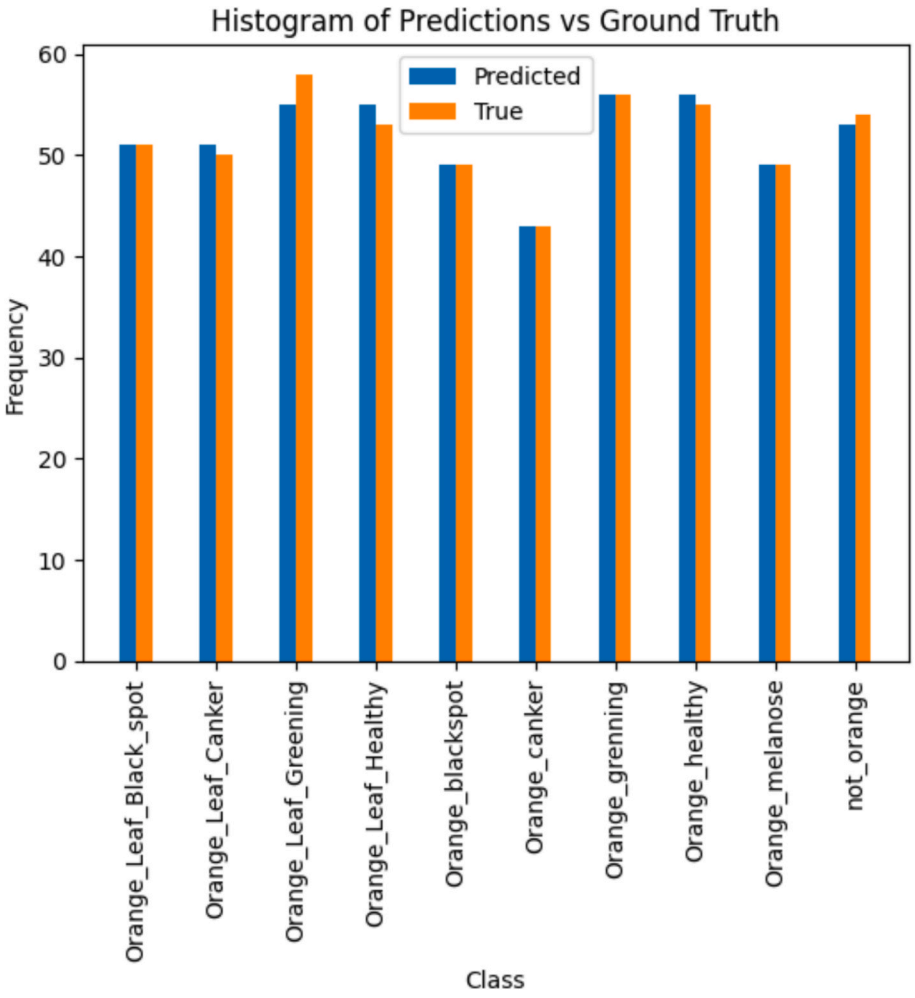


Fig. 17. Histogram of correct vs incorrect predictions for orange fruit disease images for the proposed new model.

The proposed design for the appearance of the application has three screens: menu, camera, and start. These screens require a management screen to manage communication between these three screens. First, we introduced the screens to the management screen. A menu screen is designed. This includes a logo element at the top of the menu page, a suggested application introduction text and an image-view window, a button for the camera icon, a button for the gallery icon, and a button to start. One of these three modes can be used to arrange the screens. box layout, float lay out, grid layout, we used float layout because we can use widgets in any desired position. We use the size_hint feature for the size ratio of the widgets to the image. We used the PoS_hint feature for the location of the widgets on the main image. We also used the md_bg_color feature as the background color. We used the Text feature to write the button text and label text. For text size, we used the font_size feature. In the image widget, we use the source feature to introduce the address of the image. With the ID feature, we provide a unique identifier for the desired widgets. With this identifier, we were able to access the widgets in Python code. We found and change the value of the desired feature. Buttons have an event onpress property, which is called when the button is clicked. Then, for the on_press feature and the camera icon button, we call the camera_on function. We call the move command from menuScreen to cameraScreen. The camera_on function is defined in the main.py file. that turns on a mobile phone camera or webcam. Then, for the on_press feature and gallery icon button, we call the pick_image function. This function enters the filechoser. The desired photo can then be chosen. Then, we called the start function for the on_press feature and the start button. We call the move command from menuScreen to startScreen. The start function processes the selected photo, detects the type of disease in the orange fruit, and informs the user. We used a box layout to arrange the widgets on the camera screen and placed the camera widget, camera icon, and back icon. Then, for the on_press feature, for the camera icon button, we called the capture function and the camera_off function, and changed the current screen to menuScreen. The capture function takes a photo and updates the image widget on the menu screen (Ayllon et al., 2019). The camera-off function turns off the camera. Then, for the on_pres feature and for the back icon button, we called the camera_off function and changed the current screen to menuScreen. We used a floatlayout to arrange the StartScreen, which includes a logo element at the top of the screen, a window to display an image and text for recognition, and a back button. Then, for the on-press

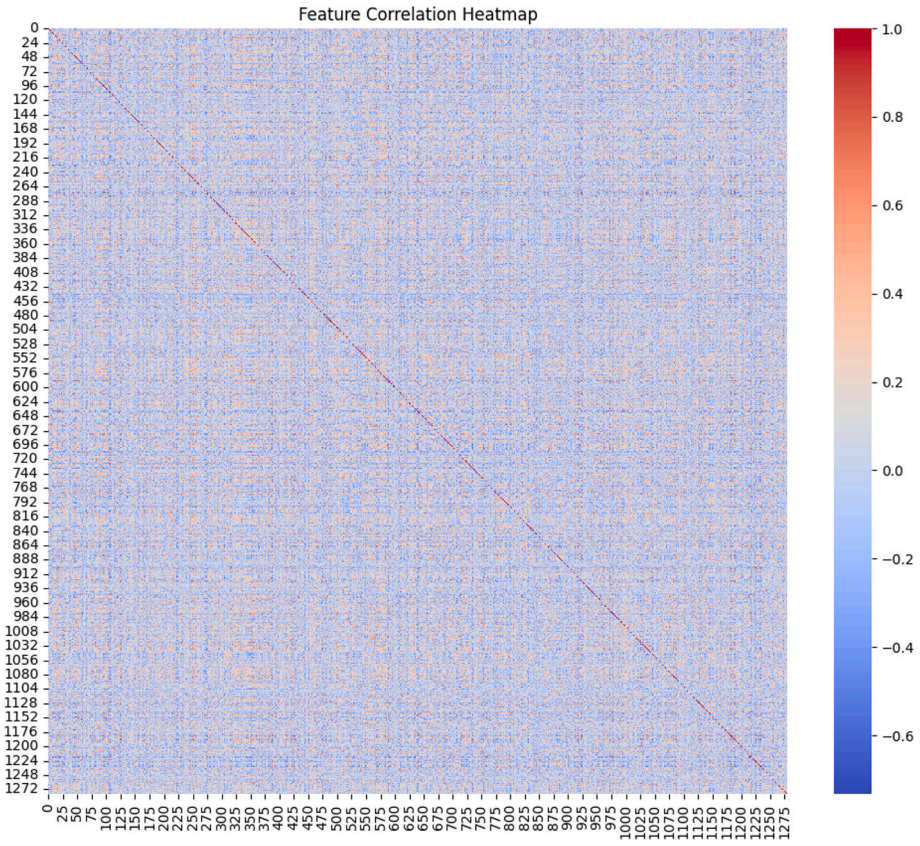


Fig. 18. Heatmap of the correlation of the proposed new model.

Table 2
Evaluation Metrics for Each Fold in 5-Fold Cross-Validation.

Fold	Accuracy	Loss	Precision	Recall	Val_Accuracy	Val_Loss	Val_Precision	Val_Recall
Fold_1	0.9898	0.029	0.9907	0.9888	0.9735	0.0835	0.9754	0.9715
Fold_2	0.9902	0.0272	0.9907	0.99	0.9774	0.0746	0.9783	0.9745
Fold_3	0.9884	0.037	0.9897	0.9878	0.9588	0.1304	0.9625	0.9568
Fold_4	0.9905	0.027	0.9905	0.9897	0.9764	0.0694	0.9774	0.9745
Fold_5	0.9867	0.0354	0.9893	0.9867	0.9749	0.0834	0.9778	0.9729

Table 3
Mean and Standard Deviation of Evaluation Metrics (Cross-Validation).

Metric	Mean ± Std
Mean Accuracy	0.9891 ± 0.0014
Mean Loss	0.0311 ± 0.0042
Mean Precision	0.9902 ± 0.0006
Mean Recall	0.9886 ± 0.0012
Mean Val_Accuracy	0.9722 ± 0.0068
Mean Val_Loss	0.0883 ± 0.0217
Mean Val_Precision	0.9743 ± 0.0060
Mean Val_Recall	0.9700 ± 0.0067

property, for the back button, we called menuScreen to change the current screen.

e) Kivy Framework for Android.

In this section, we present an algorithm for creating a graphical program using the Kivy framework to detect the type of disease of orange tree fruit and leaves for Android. This process is shown in Algorithm 3. The execution code of the proposed program in the main.py file is as follows.

Algorithm 3

Import necessary libraries **and** modules **from** Kivy, KivyMD, OpenCV, NumPy, **and** Plyer
Define global variables **for** file paths **and** image path
Initialize TensorFlow Lite interpreter **with** the model path **and allocate** tensors
Define screen classes **for** the menu, camera, **and** start screen
Define a screen manager class that inherits **from both** ScreenManager **and** MDApp
Implement methods **for** requesting permissions, turning the camera on **and** off, capturing images, starting analysis, picking images, **and** handling file selection events
Load the Kivy **language** file
Define the main application **class** inheriting **from** MDApp
Build the application **by** returning the **loaded** Kivy **language** file
Run the application **if** executed directly

To create a base for the Kivy applications, we imported the MDApp class from the Kivymdapp library (KivyMD, 2024). To upload the kv files, we imported the Builder class from the Kivy Lang library (Paymode et al., 2021). To set up the display window, we imported the window class from the KivyCore window library. To create a multiple-screen app, we imported the screen lManager class and the screen from Kivy Uix. screenmanager library. To identify the operating system, we imported the filechooser class from the Plyer library. We imported the interpreter

Table 4
Comparison of the proposed CNN model with three other models.

Model	Accuracy	Loss	Val_Accuracy	Val_Loss	Precision	Recall	Val_Precision	Val_Recall	F1_Score	Size (MB)
NewModel	0.9829	0.0548	0.9742	0.0772	0.985	0.9817	0.9771	0.9732	0.9789	8.66
MobileNetV2	0.9747	0.0698	0.9704	0.1478	0.9778	0.9712	0.954	0.9474	0.9494	8.61
MobileNetV3Large	0.973	0.0818	0.9772	0.091	0.9771	0.969	0.9781	0.9742	0.9746	11.3
NASNetMobile	0.9783	0.0644	0.9415	0.1668	0.9785	0.9739	0.9514	0.9325	0.9474	16.3

class from the tflite_runtime.interpreter library, which is a TensorFlow lite model interpreter. We imported the preview class from the camera4kivy library so that we could use the camera, and imported the os library to execute the operating system commands. The cv2 library was then imported for image processing. We imported the np class from the Numpy library to work with Arrays (Kumar et al., 2021). We imported the autoclass function from the pyobjus library. Then we loaded tensorflow lite model. To optimize the inference and assign tensors, we used the allocation_tensor function. Also, the function get_input_details receives the details of the input tensor of the model and get_output_details receives the details of the tensor output. We defined the menuScreen, cameraAscreen and StartScreen classes and defined the Manager class to manage the other three screens. We wrote the required functions of the proposed application in the manager class. The_init_method is used to initialize the program.

The camera-off method disconnects the camera module, and the camera-on method connects the camera module. We introduced the capture-path method for filepath_callback. Inside the method, we check if the platform is Android, and then we enter the permission class and the request_permissions function from the Android permission library. that the request_permission function declares the request for read and write access to the external memory. If the platform is an iOS, we call the request_ios_permissions function. We defined a capture method that first saves photos. Then, the widget image with ID img in menuScreen updates the source attribute to the new image. We defined the start method and received the source-widget image from the menu screen. We moved it to widget the image on the StartScreen. The images were read using the cv2 library. We then adjusted the input tensor value with the photo using the set_tensor function. Using the get_tensor function, we obtained the prediction of the model. With the argmax function, we obtained the highest prediction percentage from the diagnoses of canker, melanosis, black spot, greening, absence of oranges, and healthy oranges. In the lable widget In StartScreen, we set the text attribute to the prediction result for orange fruit disease. Using the pick_image method, the filechooser. open_file was opened. Using the on_file_selection method, we obtained the address of the photo selected by the filechooser, and set the source image in menuScreen with this address. The event method was used for the back key. Then, we loaded the mainkv.kv file, which is the

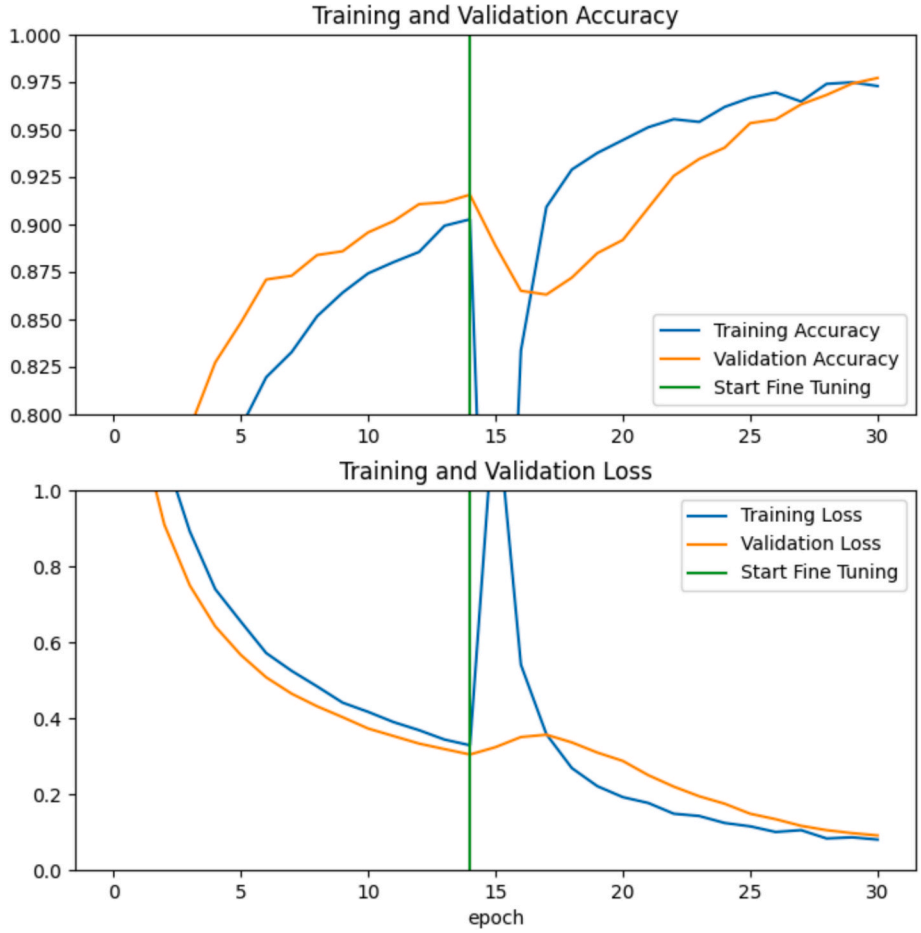


Fig. 19. Transfer learning and find-tuning diagrams of the MobileNetV3Large model.

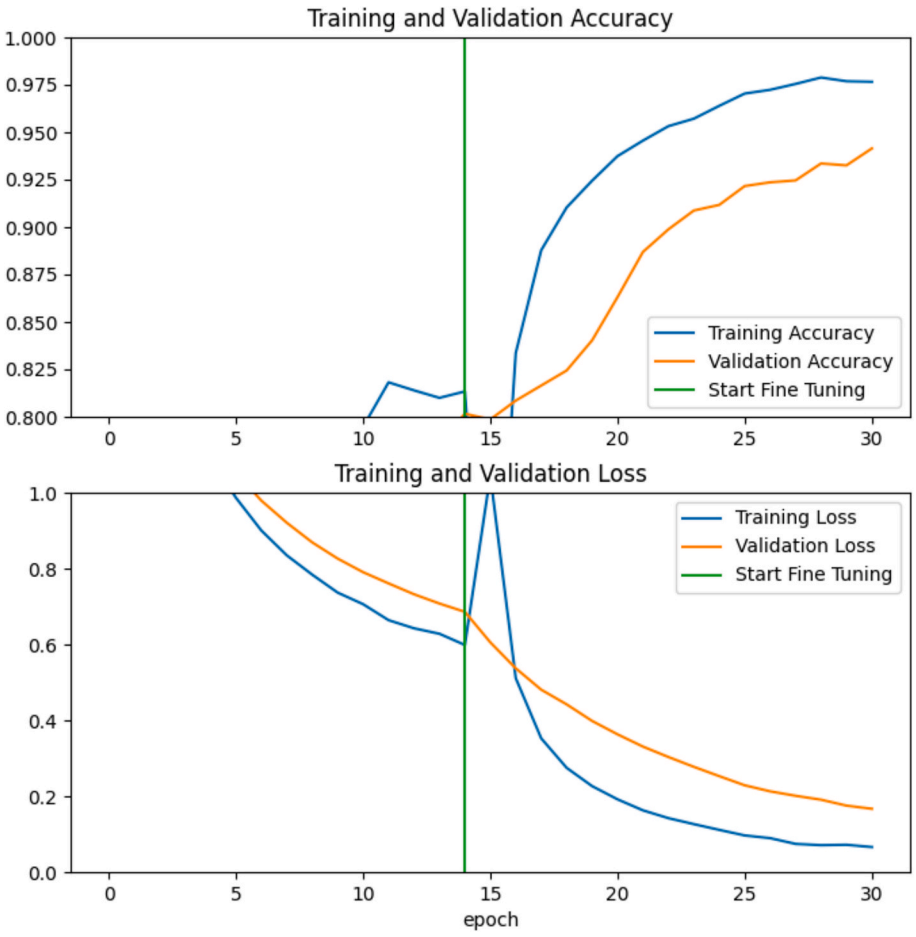


Fig. 20. Transfer learning and find-tuning diagrams of the NASNetMobile model.

graphic design file, using the builder.load_file function. The myApp class is the main class of the proposed application. Inside the build method, we returned the kv variable.

Finally, we ran () myApp class. This program can be run on Windows, Linux, and Mac OS. To run it on Android, we converted it to an APK package using the python-for-android library and Buildozer tool. We will explain this further. We use to run it on ios, we used the Buildozer tool.

f) Kivy language for Raspberry pi and PC.

The proposed program can be run on Raspberry Pi and PC and has a graphical environment. We used kivy framework and kv language. The process is presented in Algorithm 4. The codes of the proposed application design with kv language in mainkv.kv file are as follows.

Algorithm 4

Manager:
MenuScreen
StartScreen
MenuScreen:
Layout:
Background: white
Header: "This application can detect the type of Orange Fruit Disease by ARMAN FOROUGHI"
Image: "orange.png" (top)
Main Area: Image ("orange.png")
Bottom Controls:
Email:
Button: "Set Email" –> app.set_email()
TextField: "Enter your Email"
Time:

(*continued on next column*)

(*continued*)

Algorithm 4

Button: "Set Time" –> app.set_time()
TextField: "Enter start hour"
Button: "Start" –> switch **to** "StartScreen" **and** call app.camera_on()
StartScreen:
Layout:
Images: "please_wait.jpg" (several images displayed)
Labels: Display image identifiers
Preview: Aspect ratio '16:9'
Button: "Back" –> switch **to** "MenuScreen" **and** call app.camera_off()

The proposed design for the application has two screens: a menu and a start. These screens require a management screen to manage the communication between them. First, we introduced the screens to the management screen. It includes a logo element at the top of the menu screen, a text introducing the proposed program and an image-viewing window, a button to set the email, a button to set the clock, and a button to start. we used float layout. because widgets can be used in any desired position. Buttons have an event on_press property, which is called when the button is clicked. For the on_press property and email set button, we call the set_email function. The set_email function is defined in the main.py file. Then we call the set_time function for the on_press feature and the clock setting button. For the on_press feature of the start button, we changed the current screen to startScreen and called the camera_on function. The camera_on function activates the Raspberry Pi or computer camera.On the start screen, there is a plate for the Raspberry Pi camera and five plates for the photos taken. The operation of taking a photo and recognizing it using the newly created model and sending an email to the user is performed every 24 h. A back button was also placed

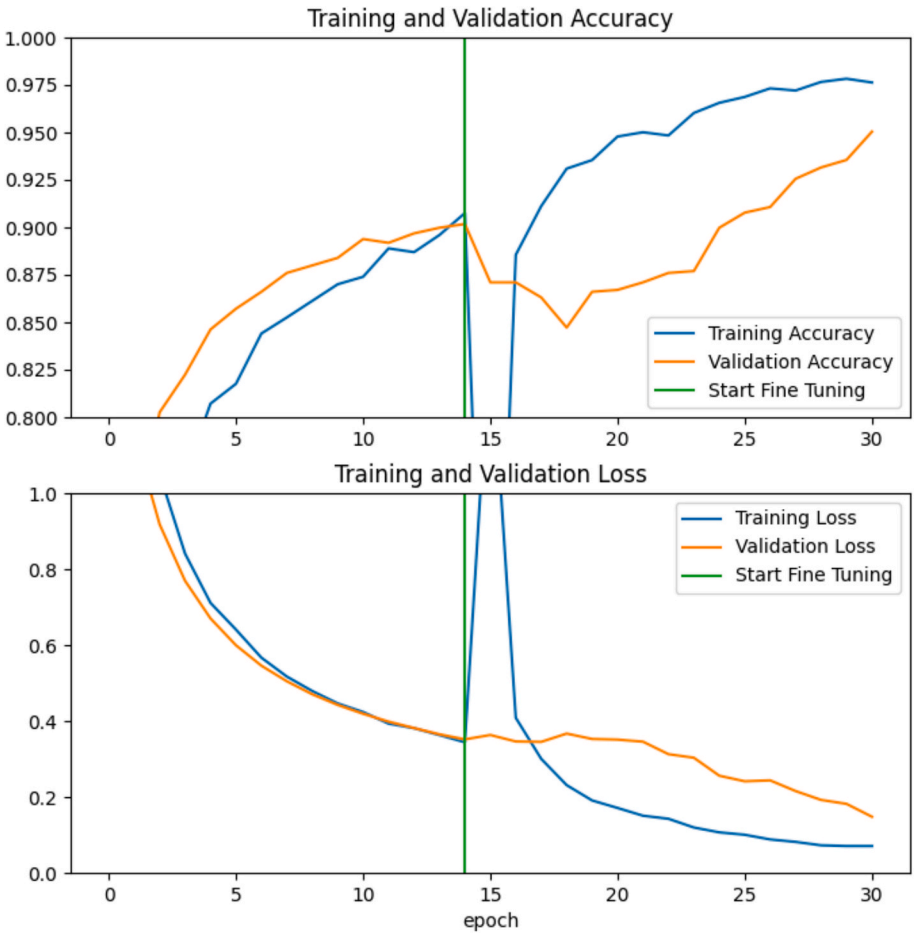


Fig. 21. Transfer learning and find-tuning diagrams of the MobileNetV2 model.

to return to the main page.

g) Kivy farmework for Raspberry Pi and Pc.

In this section, we present an algorithm for creating a graphical program using the Kivy framework to detect the type of disease of orange tree fruit and leaves for Raspberry Pi and PC. This process is shown in Algorithm 5. The execution code of the proposed program in the main.py file is as follows.

Algorithm 5

Import *necessary libraries* ***and*** *modules* ***from*** *Kivy, KivyMD, OpenCV, NumPy,* ***and*** *Plyer*
Define global variables **for** File paths for images and model, Email address, Hour for starting photo capture
Initialize TensorFlow Lite interpreter **with** the model path **and allocate** tensors
Define screen classes **for** the menu **and** start screen
Define a screen manager class that inherits **from both** ScreenManager **and** MDApp
Implement methods **for** send_mail_function, set_email, set_time, start_camera(At specified hour), turning the camera on **and** off, capturing images, start_detect, **and** handling file selection events
Load the Kivy **language** file
Define the main application **class** inheriting **from** MDApp
Build the application **by** returning the **loaded** Kivy **language** file
Run the application **if** executed directly

The relevant libraries have been imported. In addition, the kivy.clock library was imported for scheduling functions and the smtplib library was imported for sending emails. The function send_mail_function is written to send an email. In addition, the set_email function sets the user's email, and the set_time function sets the start time of photography and the operation for detecting the type of disease of the fruit and leaves of the orange tree. The camera_off function disabled the camera. The capture function sets a captured photo on the second page of the program. The start_detect function opens photos captured by the camera. It detects the type of disease using the proposed deep learning model. and writes his diagnosis under the photo and sends an email containing the name of the photo and diagnosis to the user. The start_camera function first checks that the system clock is the same as the clock set by the user. And the current screen checks to be startScreen. A for-loop is written to repeat calling the capture function and the start_detect function five times; then, the Clock.schedule_interval function is called to execute the start_camera function every hour.

h) Building an APK using a Buildozer.

The entire construction process was automated using a Buildozer. The Python-for-android development tool downloads and compiles prerequisites, including the Android SDK, NDK, and related libraries, and then creates an apk that can be installed and run on Android devices. To use the camera on our Android device, we used the camera4kivy library. We used the Opencv library in the mainpy file to open the images. To convert the application to Android, we need an opencv SDK for Android. The construction process using a Buildozer took approximately 45 min.

4. Results

We tested the trained model of the proposed program on 5073 images and compared its recognition with image labels. The results are shown in Fig. 9. The diagnosis of fruit and leaf disease of the orange tree is shown on the Y-axis and image labels on the X-axis (Kaggle, n.d.). we

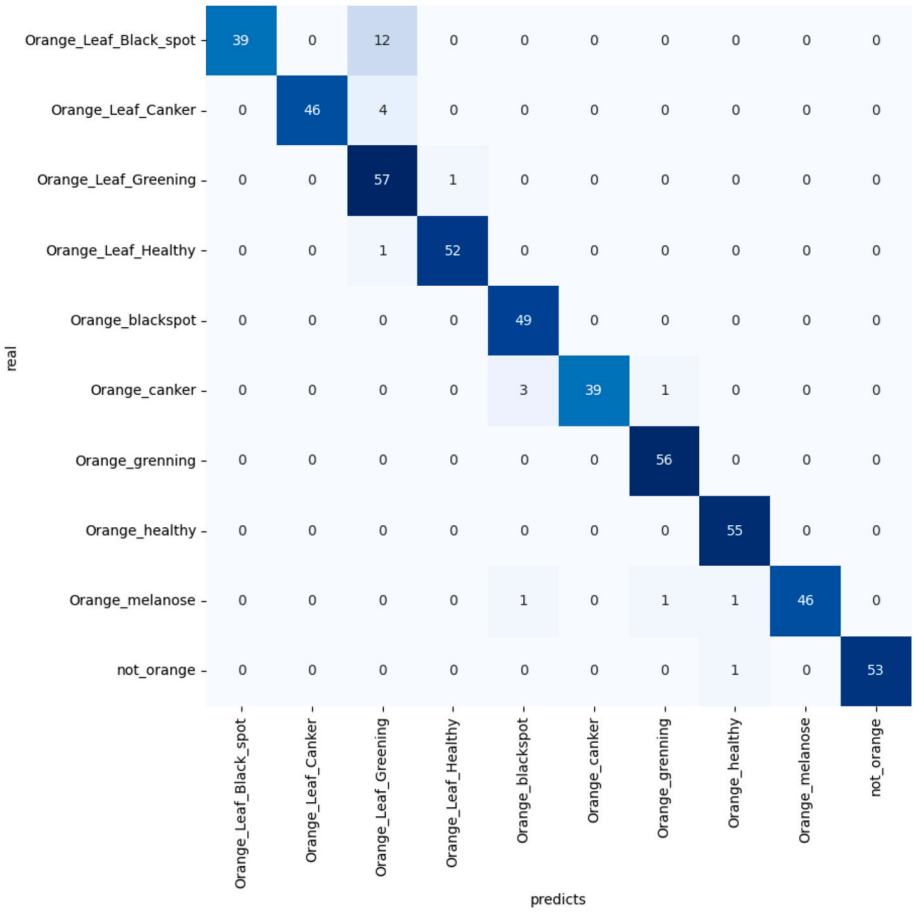


Fig. 22. Confusion matrix of the MobileNetV2 model.

checked the APK output of the proposed application on the Android 13 device. Then, the orange tree orchard was photographed with the Android device, and the application declared the fruit and leaves of the orange tree as healthy, as shown in Fig. 10. A custom dataset was prepared using images collected from orchards in the vicinity of Valencia city. The experiments in this study were conducted on these images, and Fig. 10 presents a sample from the dataset. In addition, as shown in Fig. 11, a photo of orange fruit black spot disease was selected from the gallery of the Android device, which the application correctly declared as a black spot disease. In addition, using an Android operating system mobile device, an apple fruit was photographed and declared non-orange, as shown in Fig. 12. This is the main screen for Raspberry Pi and PC applications, as shown in Fig. 13. in which a section was designed to allow the user to set an email. In the second part, he sets the start time of the program. After that, by pressing the start button of the application, he enters another page, where the operation of taking pictures, diagnosing the disease, and notifying the user by email every 24 h. Fig. 14 shows the performance of the program in the Linux environment on the Raspberry Pi device. A program whose algorithm and flowchart were explained was implemented on Raspberry Pi. The Raspberry Pi was connected to the camera module, also the Raspberry Pi was tested with the camera module in the gardens around Valencia. After the Raspberry Pi device took a photo of the fruit and leaf of the orange tree, the health of the fruit and leaf was displayed, and the text containing the photo number and the corresponding diagnosis was sent to us via email.

5. Discussion

a) Review of the proposed application.

Diagnosing fruit and leaf diseases is important for farmers. First, a dataset of fruit and leaf disease images of orange trees were collected from the Kaggle website. (Storey et al., 2022). At first, we built a deep learning model by combining HSL color model and MobileNetV2 model. which has a more optimal diagnosis compared to other deep learning models. Then, using the Kivy framework, we designed a graphics application in Python that can take photos with the camera or select photos from the gallery. Then it can detect the diseases of orange tree fruit and leaves from the selected image (Sharif et al., 2018). Next, we built a CNN model using transfer learning. In this method, we loaded the MobileNetV2 model with ImageNet weights, without Top layer. Subsequently, we froze the weights. We created a new top layer and a bottom layer to convert RGB to HSL. We designed the proposed model according to eight classes of orange fruit and leaf diseases, one orange fruit and leaf health class, and one non-orange class. Then the created model was compiled and trained. Then we fine-tuned this model. At this stage, the frozen weights were removed from the frozen state. This model was retrained with a lower learning rate. Then we converted the trained

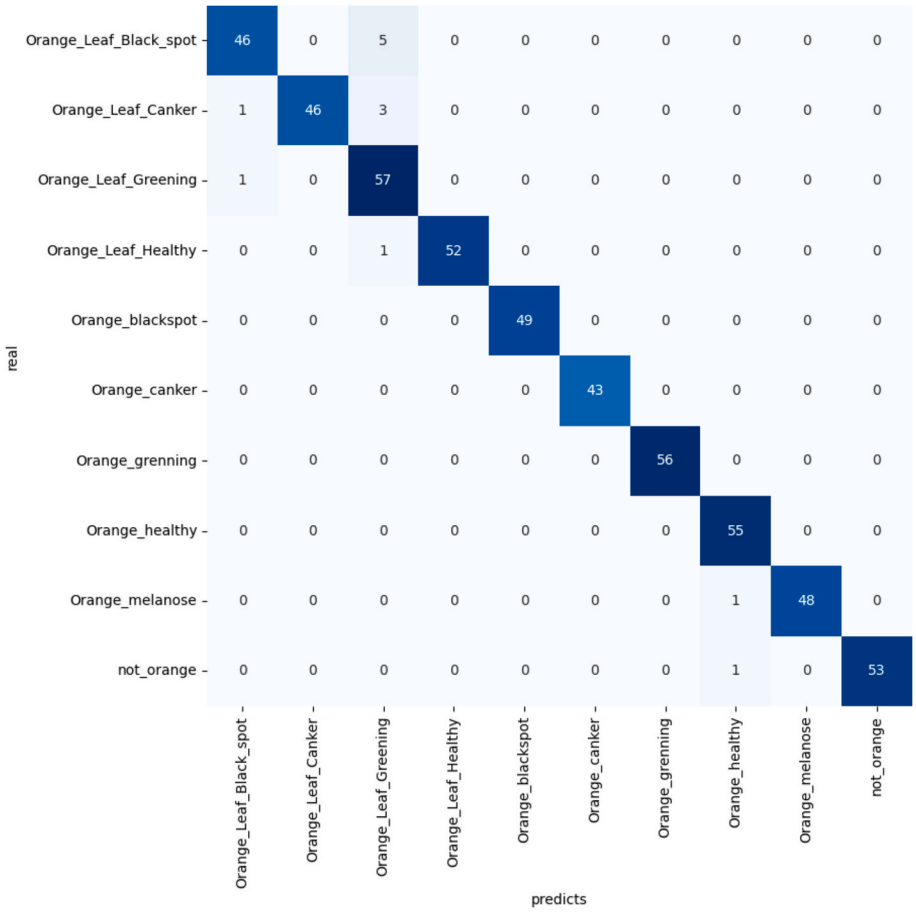


Fig. 23. Confusion matrix of the MobileNetV3Large model.

model to tflite model. We then saved it for use in an application created with the Kivy framework. The proposed program can be run on Windows, Linux and Mac operating systems. We also output it to run on Android and iOS using Buildozer. A comparison of these studies is shown in Table 1. In the proposed program, the detection time of orange tree fruit and leaf disease is 0.13 s. Compared to the studies in Table 1, it can be concluded that the application of the proposed model provides the answer in a much shorter time. They used a server to detect fruit or plant leaf disease. In addition, their application requires the Internet to connect to the server and upload photos from the mobile phone to the server. Therefore, the detection process required approximately 10 s. This is one of the limitations of previous articles. In this study we solved this problem for farmers. that farmers can diagnose orange fruit diseases offline. Also, for the use of farmers in orange orchards, a separate program was designed for Raspberry Pi. In this graphic program, the farmer can set his email so that the result of the diagnosis of the disease or the health of the fruit and leaves of the orange tree will be sent to him daily through the Internet. This application and new model can be used in large gardens that require CCTV cameras. This program is very useful for farmers. In previous studies, it was found that the proposed model does not convert to TensorFlow lite. In the proposed application, we used TensorFlow lite, which can be used on mobile devices and does not require a separate server. In addition, no internet or cloud space is required. Fig. 15 corresponds to transfer learning where the model weights are frozen and only the top layer is trained. In this figure, the upper graph shows the amount of learning and the increase in accuracy of the model in fifteen periods on TRAINING and VALIDATION data, which has reached 89.15 percent accuracy. The lower graph shows the learning loss in fifteen periods, which reached 0.3558. Fig. 16 shows the fine-tuning of our model after transfer learning. The weights are not frozen and are trained with a low learning rate. In this figure, the upper graph shows the accuracy of the model, which has reached an accuracy of 98.73 percent. The bottom chart shows the LOSS figure of the model in each period, which reached 0.0402. The right part of the learning transfer diagram and the left part of the model fine-tuning diagram. In Fig. 17, the histogram of the predicted and ground truth values of the proposed new model is shown. Fig. 18 shows the heatmap of the correlation of the extracted features from the last layer of the proposed new model. The correlation between the specified samples indicates that the proposed new model has high feature consistency.

Table 2 shows the performance of the proposed new model in each of the five folds of the cross-validation process. The results indicate that the proposed model performs consistently across all five folds, suggesting that it has achieved balanced learning. The metrics presented include Accuracy, Loss, Precision, and Recall for both the training and validation datasets in each fold. Table 3 shows the mean and standard deviation of the evaluation metrics for the proposed new model during the 5-fold cross validation process. The proposed model demonstrates high accuracy, low standard deviation, and reliable performance across the five folds. The evaluated metrics include Accuracy, Loss, Precision, and Recall for both the training and validation datasets.

b) Comparison of selected CNN models with three other models.

For the optimal use of the application in Android systems, a low volume of the application is required. Also, the accuracy of the application is very important to diagnose the type of disease of the fruit and leaves of the orange tree. We compared and analyzed the built model with three other models named MobileNetV3Large, NASNetMobile, MobileNetV2 and the results are shown in Table 2. It is smaller and more

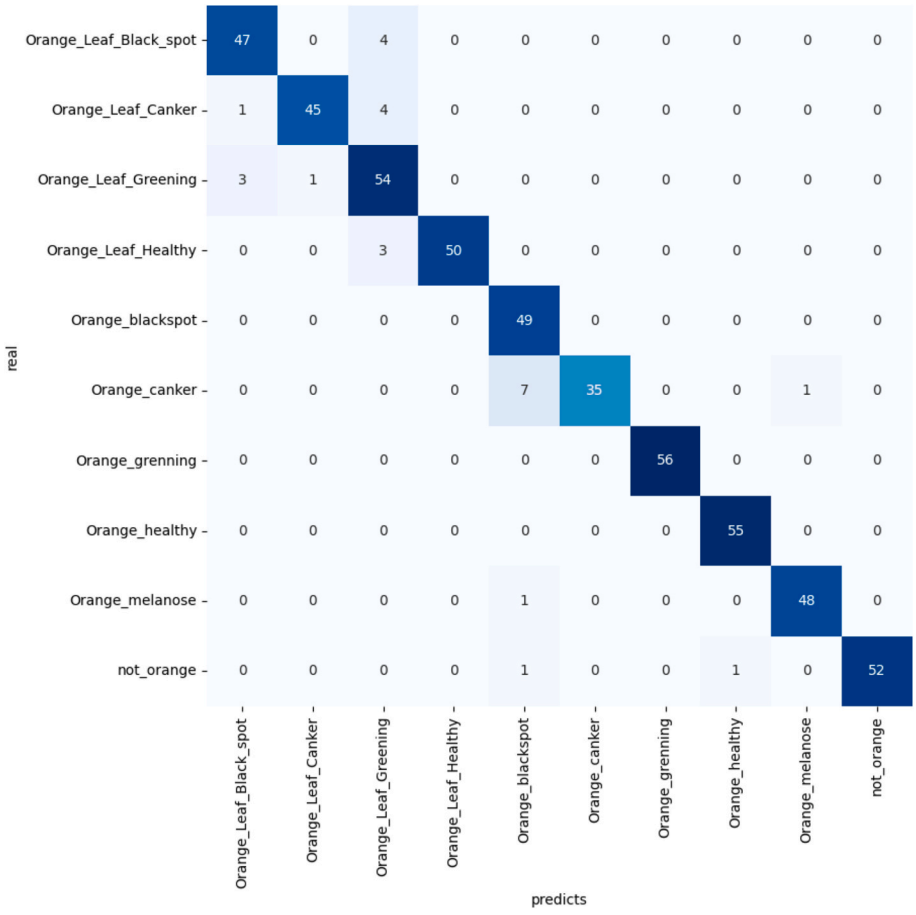


Fig. 24. Confusion matrix of the NASNetMobile model.

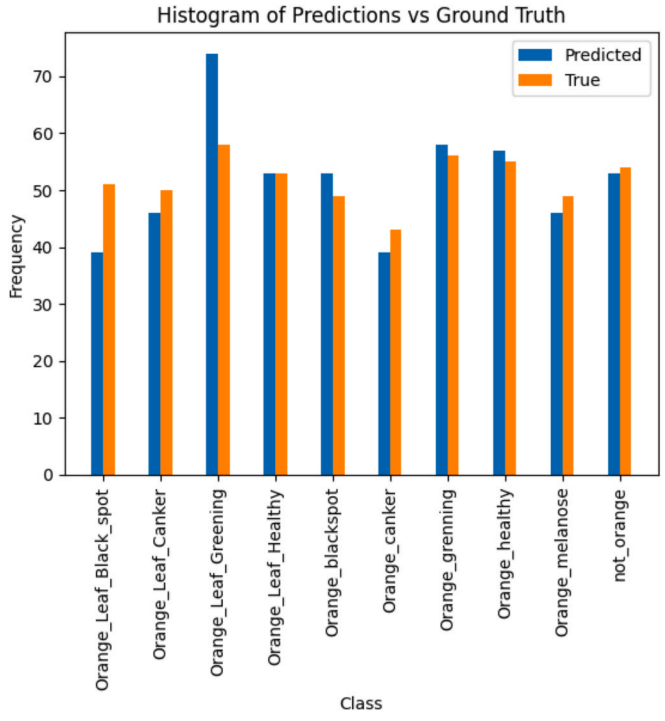


Fig. 25. Histogram of correct vs incorrect predictions for orange fruit disease images for the MobileNetV2 model.

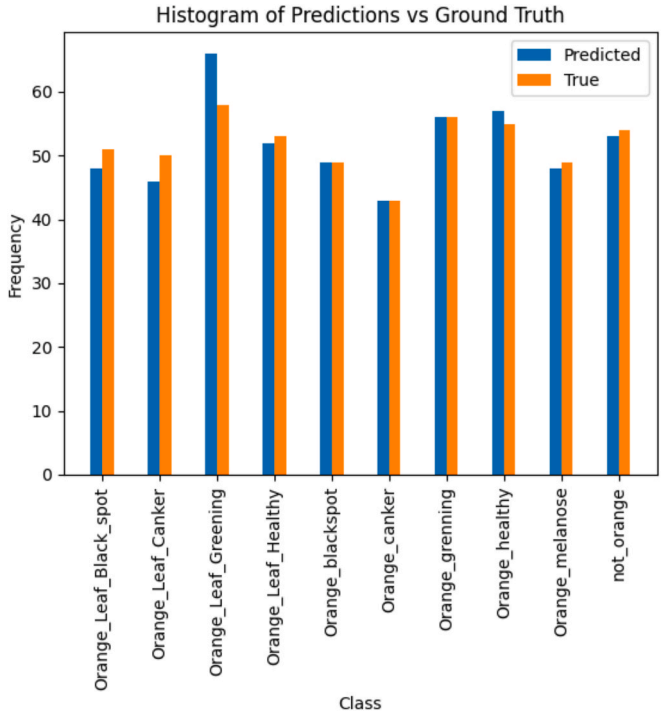


Fig. 26. Histogram of correct vs incorrect predictions for orange fruit disease images for the MobileNetV3Large model.

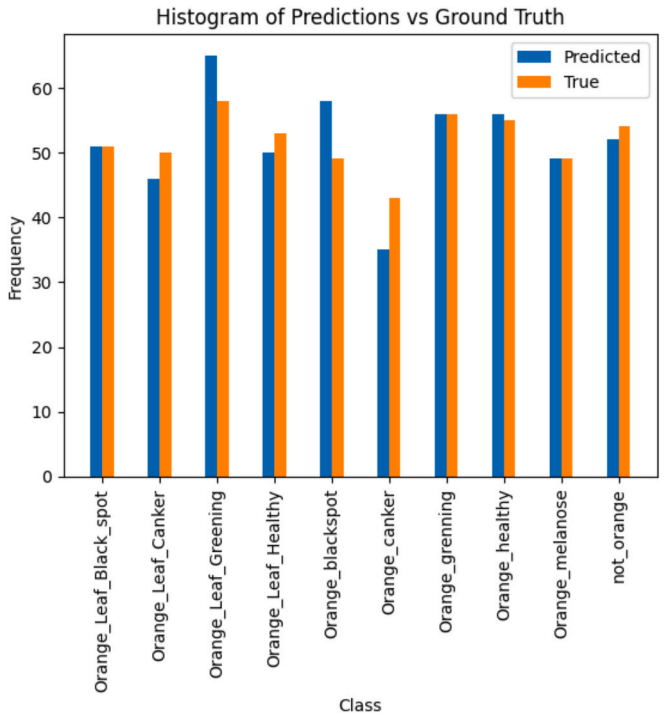


Fig. 27. Histogram of correct vs incorrect predictions for orange fruit disease images for the NASNetMobile model.

accurate than other models. Table 4 shows the size (MB) is the output size of the model, and the loss and accuracy of the models are compared. The new model has smaller output size, lower loss and higher accuracy and better detection.

The transfer learning and fine-tuning diagrams of MobileNetV3Large are shown in Fig. 19. In these figures, the training accuracy is shown by a blue line, the validation accuracy is shown by an orange line, and the start fine-tuning is shown by a green line. In these figures, it is clear that when fine-tuning starts, the accuracy increases, and the error is greatly reduced. Transfer learning and find tuning diagrams of NASNetMobile are shown in Fig. 20. Transfer learning and find tuning diagrams of MobileNetV2 are shown in Fig. 21.

In Fig. 22, the confusion matrix of the MobileNetV2 model is shown. As shown in the figure, this model mispredicted 26 images. Fig. 23 displays the confusion matrix of the MobileNetV3Large model, where 13 images were incorrectly predicted. In Fig. 24, the confusion matrix of the NASNetMobile model is shown, with 27 mispredicted images. By comparing these results with the proposed new model, which only mispredicted 11 images, it can be concluded that the proposed model achieves higher accuracy.

In Fig. 25, the histogram of the predicted and ground truth values for the MobileNetV2 model is shown. In Fig. 26, the histogram of the predicted and ground truth values for the MobileNetV3Large model is shown. In Fig. 27, the histogram of the predicted and ground truth values for the NASNetMobile model is shown. In each histogram, the blue column represents the predicted labels, while the orange-colored column represents the true labels. The X-axis shows the disease classes of the fruit and leaves of the orange tree, and the Y-axis represents the number of labeled samples.

Fig. 28 shows the heatmap of the correlation of the extracted features from the last layer of the MobileNetV2 model. Fig. 29 shows the heatmap of the correlation of the extracted features from the last layer of the MobileNetV3Large model. Fig. 30 shows the heatmap of the correlation of the extracted features from the last layer of the NASNetMobile model. In the comparison of the three models, MobileNetV2, MobileNetV3Large, and NASNetMobile with the proposed new model, the

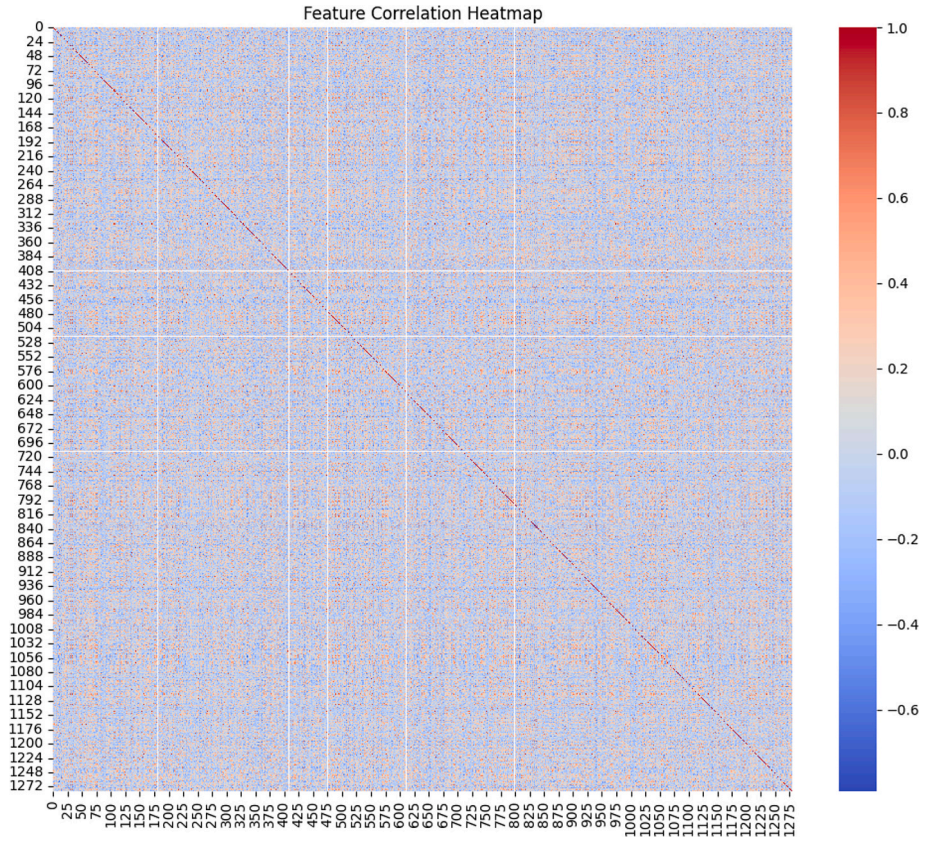


Fig. 28. Heatmap of the correlation of the extracted features from the last layer of the MobileNetV2 model.

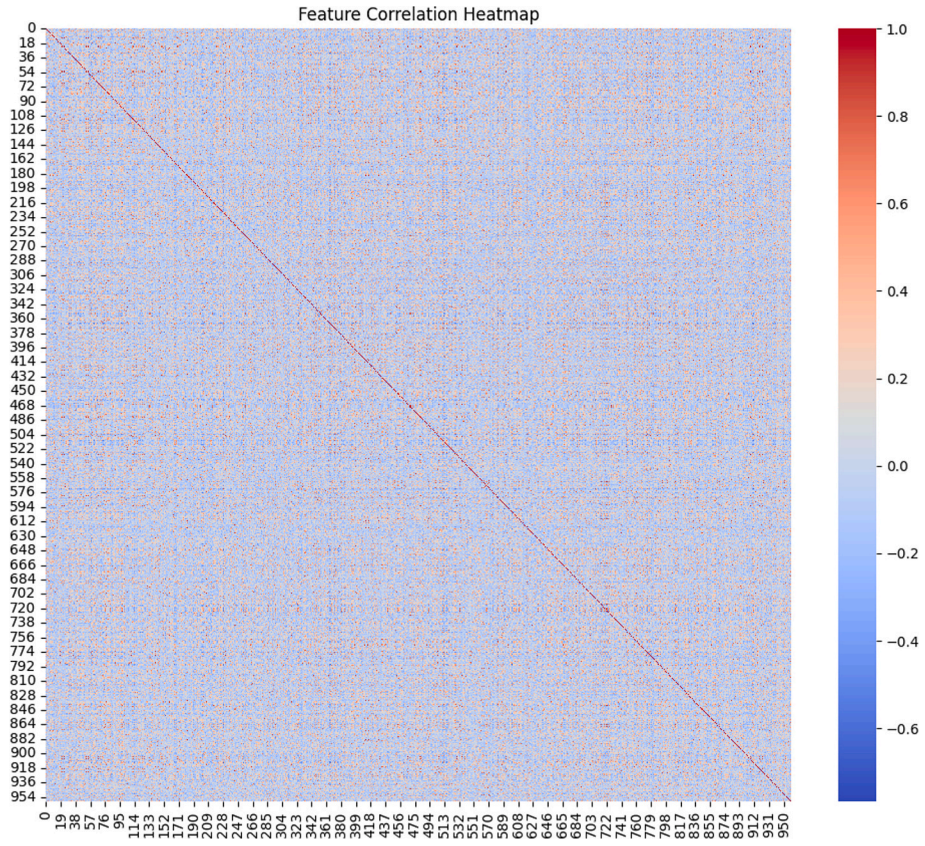


Fig. 29. Heatmap of the correlation of the extracted features from the last layer of the MobileNetV3Large model.

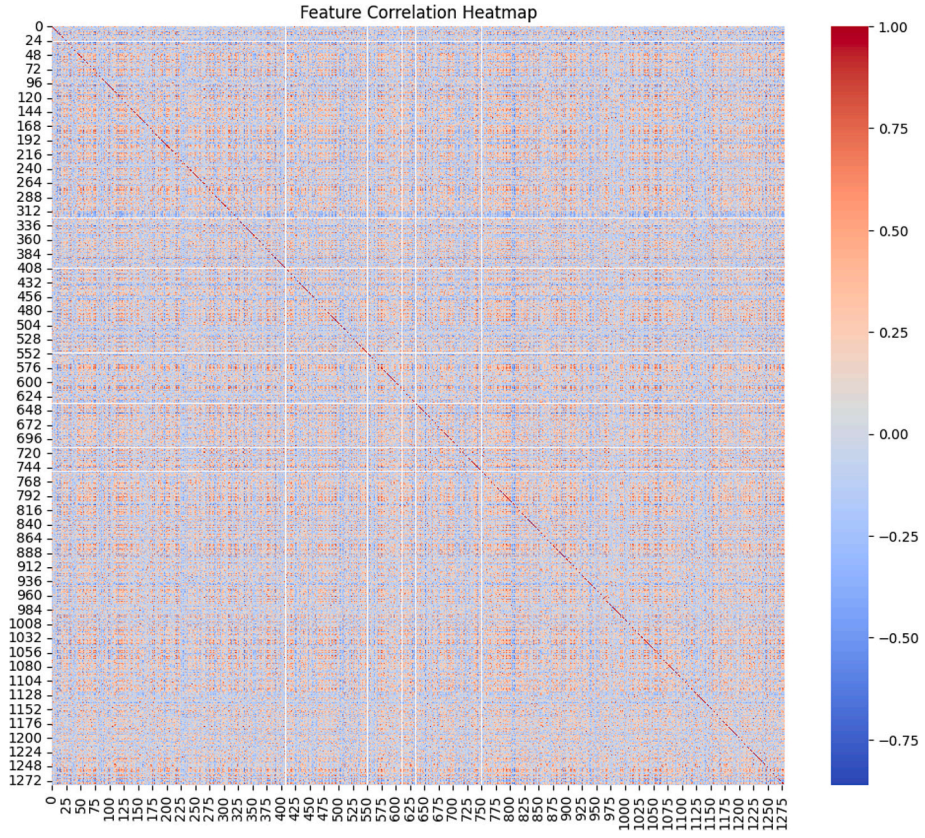


Fig. 30. Heatmap of the correlation of the extracted features from the last layer of the NASNetMobile model.

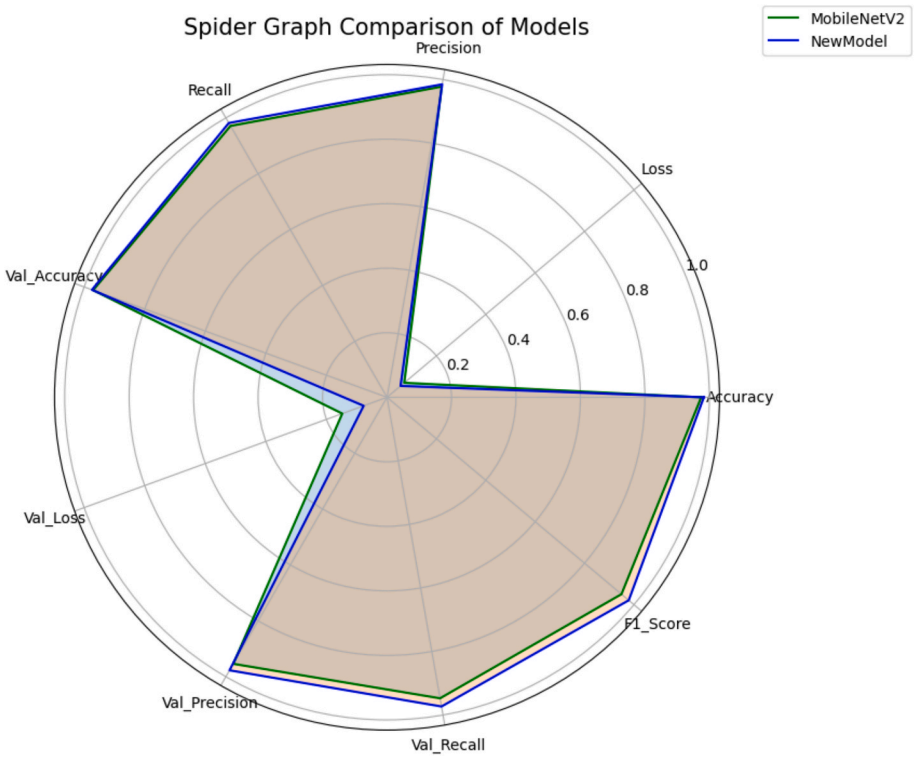


Fig. 31. Spider graph comparing the MobileNetV2 model with the proposed new model.

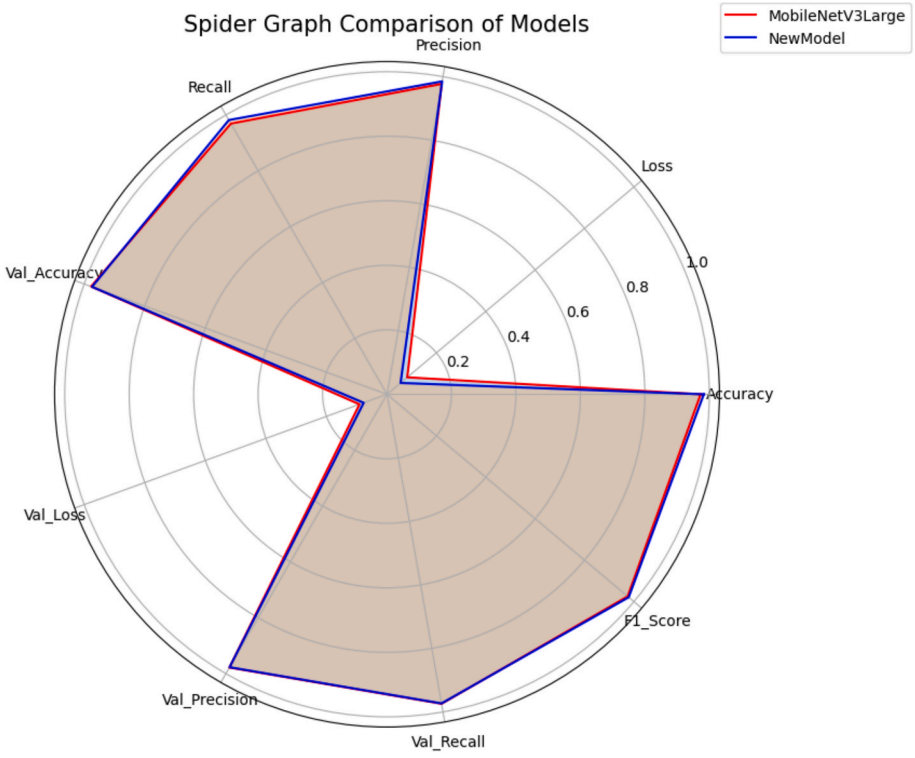


Fig. 32. Spider graph comparing the MobileNetV3Large model with the proposed new model.

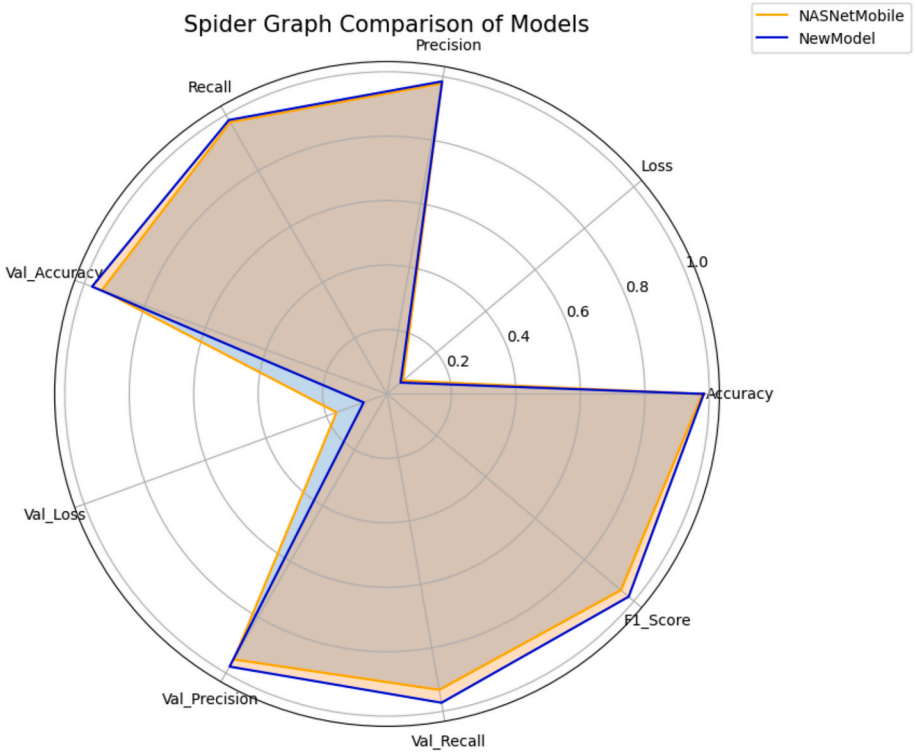


Fig. 33. Spider graph comparing the NASNetMobile model with the proposed new model.

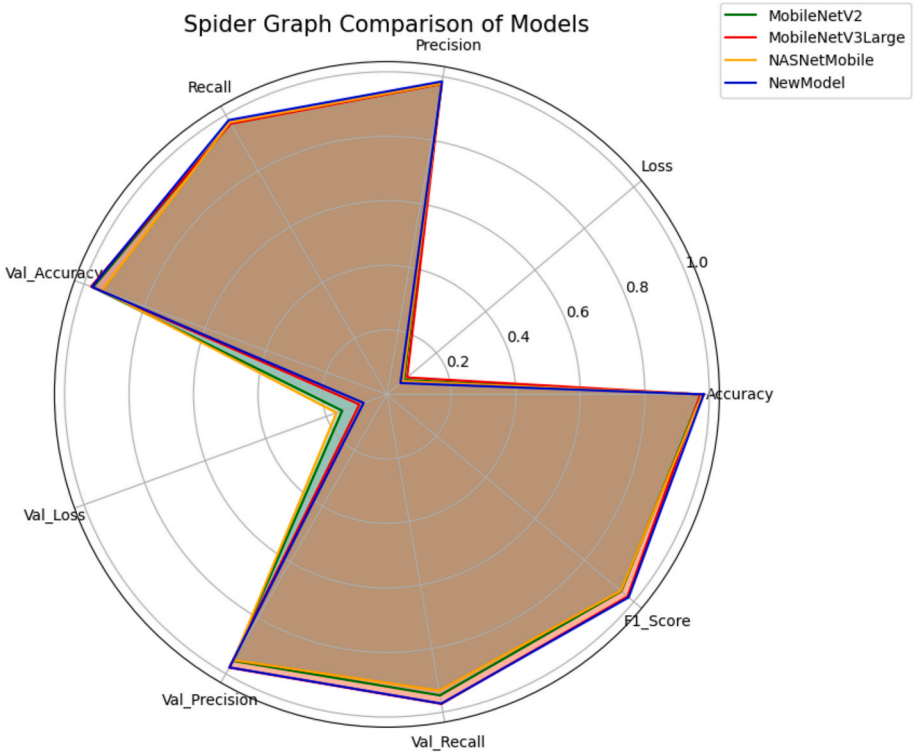


Fig. 34. Spider graph comparing MobileNetV2, MobileNetV3Large, NASNetMobile, and the proposed new model.

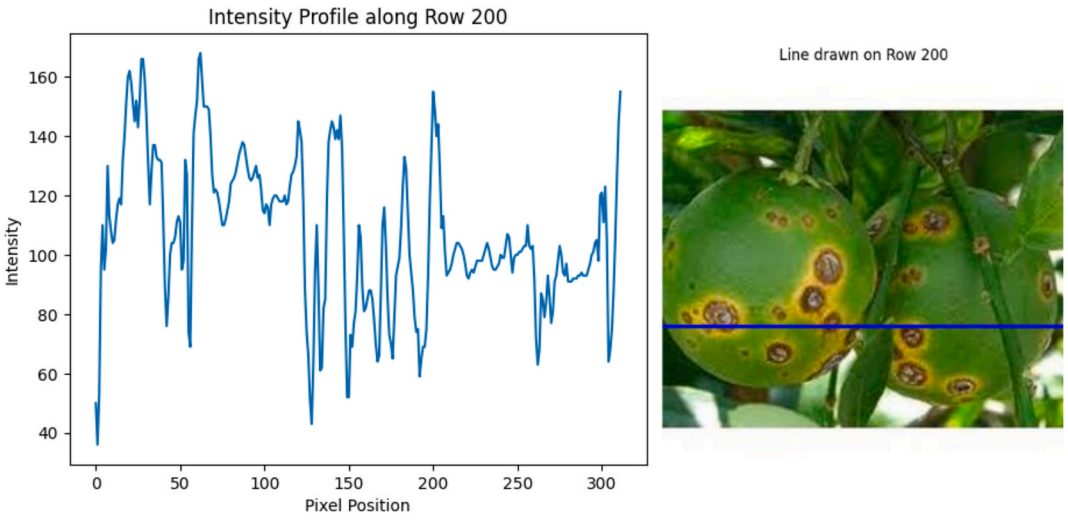


Fig. 35. Intensity profile along a selected row in the image of orange fruit canker disease, highlighting fluctuations that indicate the most infected regions.

proposed model demonstrates higher compatibility in its features.

Figs. 31, 32, and 33 show spider graphs comparing the MobileNetV2, MobileNetV3Large, and NASNetMobile models, respectively, with the proposed new model based on nine evaluation metrics: accuracy, precision, recall, F1 score, loss, recall value, loss value, precision value, and accuracy value. In Fig. 34, a spider graph is shown that compares the performance of all four models, MobileNetV2, MobileNetV3Large, NASNetMobile, and the proposed new model, based on the same metrics.

Fig. 35 shows the intensity profile along a selected row in the image of orange fruit canker disease. The selected row corresponds to the most infected area. The intensity fluctuations indicate the regions affected by the canker disease.

6. Conclusion

The proposed application can detect all types of orange fruit and leaf diseases (black spot, canker, melanosis and greening) with a response speed of 0.13 s and with an accuracy of 98.29 %. It also detects healthy oranges and leaves and declares any object except orange fruit which is non-orange. The proposed program does not require the Internet and works without a server. In offline mode, it can detect the type of disease of orange tree fruit and leaves. We developed a separate program for Raspberry Pi and computer that the farmer could use in large orange orchards. This application is connected to the Internet system and can send emails to the user and the result of diagnosing the disease of the fruit and leaves of the orange tree. In future studies, we plan to use the proposed application in the construction of smart robots and drones to diagnose various fruit diseases. In addition to diagnosing various fruit diseases, other algorithms will also be used to identify fruit disease spots. Furthermore, we plan to combine the proposed system with a fertilizer irrigation system (Aldegheishem et al., 2023) and gas sensors to identify other issues (Wang et al., 2023).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work has been funded by the “Ministerio de Ciencia e Innovación” through the Project PID2020-114467RRC33/AEI/10.13039/501100011033 and by the “Ministerio de Economía y Competitividad” through the Project TED2021-131040BC31.

Data availability

Data will be made available on request.

References

Aftab, S., Lal, C., Beejal, S. K., & Fatima, A. (2022). Raspberry Pi (Python AI) for plant disease detection. *International Journal of Current Research and Review, 14*(03), 36–42. https://doi.org/10.31782/ijcrr.2022.14307

Aldegheishem, A., Viciano-Tudela, S., Parra, L., Alrajeh, N., & Lloret, J. (2023). Proposal of a sensor node to determine the suitability of reclaimed water for green areas irrigation in smart city context. *IEEE Sensors Journal, 23*(19), 23348–23355. https://doi.org/10.1109/jsen.2023.3305073

Al-Tuwaijari, J. M., Jasim, M. A., & Raheem, M.-A.-B. (2020). Deep learning techniques toward advancement of plant leaf diseases detection. In *In 2020 2nd Al-Noor International Conference for Science and Technology (NICST)*. https://doi.org/10.1109/nicst50904.2020.9280320

Asani, E. O., Osadeyi, Y. P., Adegun, A. A., Viriri, S., Ayoola, J. A., & Kolawole, E. A. (2023). mPD-APP: A mobile-enabled plant diseases diagnosis application using convolutional neural network toward the attainment of a food secure world. *Frontiers in Artificial Intelligence, 6*. https://doi.org/10.3389/frai.2023.1227950

Asif, M. K. R., Rahman, M. A., & Hena, M. H. (2020). CNN based disease detection approach on potato leaves. In *In 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS)*. https://doi.org/10.1109/iciss49785.2020.9316021

Ayllon, M. A., Cruz, M. J., Mendoza, J. J., & Tomas, M. C. (2019). Detection of overall fruit maturity of local fruits using convolutional neural networks through image processing. In *In Proceedings of the 2nd International Conference on Computing and Big Data*. https://doi.org/10.1145/3366650.3366681

Barman, U., Choudhury, R. D., Sahu, D., & Barman, G. G. (2020). Comparison of convolution neural networks for smartphone image-based real-time classification of citrus leaf disease. *Computers and Electronics in Agriculture, 177*, Article 105661. https://doi.org/10.1016/j.compag.2020.105661

Batool, A., Hyder, S. B., Rahim, A., Waheed, N., Asghar, M. A., & Fawad. (2020). Classification and identification of tomato leaf disease using deep neural network. In *In 2020 International Conference on Engineering and Emerging Technologies (ICEET)*. https://doi.org/10.1109/iceet48479.2020.9048207

Buildozer. (n.d.). Welcome to Buildozer's documentation! — Buildozer 0.11 documentation. Retrieved March 27, 2024, from https://buildozer.readthedocs.io.

Faisal, S., Javed, K., Ali, S., Alasiry, A., Marzougui, M., Attique Khan, M., & Cha, J.-H. (2023). Deep transfer learning based detection and classification of citrus plant diseases. *Computers, Materials & Continua, 76*(1), 895–914. https://doi.org/10.32604/cmc.2023.039781

Foong, C. C., Meng, G. K., & Tze, L. L. (2021). Convolutional neural network based rotten fruit detection using ResNet50. In *In 2021 IEEE 12th Control and System Graduate Research Colloquium (ICSGRC)*. https://doi.org/10.1109/icsgrc53186.2021.9515280

Foroughi, A., Jimenez, J. M., & Lloret, J. (2023, October). A new image processing system to diagnose the orange fruit disease. In 2023 10th International Conference on Internet of Things: Systems, Management and Security (IOTSMS). Retrieved January 20, 2024, from https://doi.org/10.1109/iotsms59855.2023.10325781.

Gencturk, B., et al. (2023). Detection of hazelnut varieties and development of mobile application with CNN data fusion feature reduction-based models. *European Food Research and Technology*. https://doi.org/10.1007/s00217-023-04369-9

Herman, H., Cenggoro, T. W., Susanto, A., & Pardamean, B. (2021). Deep learning for oil palm fruit ripeness classification with DenseNet. In *In 2021 International Conference on Information Management and Technology (ICIMTech)*. https://doi.org/10.1109/icimtech53080.2021.9534988

ImageNet. (n.d.). ImageNet. Retrieved March 27, 2024, from https://www.image-net.org/.

Kaggle. (n.d.). Your machine learning and data science community. Retrieved March 25, 2024, from https://www.kaggle.com/.

Karar, M. E., Alsunaydi, F., Albusaymi, S., & Alotaibi, S. (2021). A new mobile application of agricultural pests recognition using deep learning in cloud computing system. *Alexandria Engineering Journal, 60*(5), 4423–4432. https://doi.org/10.1016/j.aej.2021.03.009

Kumar, P. K. V., Rao, E. G., Anitha, G., & Kumar, G. K. (2021). Plant disease detection using convolutional neural networks. In 2021 5th International Conference on Trends in Electronics and Informatics (ICOEI). Retrieved March 25, 2024, from https://doi.org/10.1109/icoei51242.2021.9453045.

Kumar, R. (2021, October). Deep learning to detect plant diseases. In 2021 6th International Conference on Signal Processing, Computing and Control (ISPCC). Retrieved March 25, 2024, from https://doi.org/10.1109/ispcc53510.2021.9609389.

KivyMD. (2024). KivyMD 2.0.1.dev0 documentation. Retrieved March 27, 2024, from https://kivymd.readthedocs.io.

Lanjewar, M. G., & Panchbhai, K. G. (2022). Convolutional neural network based tea leaf disease prediction system on smart phone using paas cloud. *Neural Computing and Applications, 35*(3), 2755–2771. https://doi.org/10.1007/s00521-022-07743-y

Malathy, S., Karthiga, R. R., Swetha, K., & Preethi, G. (2021, January). Disease detection in fruits using image processing. In 2021 6th International Conference on Inventive Computation Technologies (ICICT). Retrieved March 25, 2024, from https://doi.org/10.1109/icict50816.2021.9358541.

Manoharan, N., Thomas, V. J., & Anto Sahaya Dhas, D. (2021, August). Identification of mango leaf disease using deep learning. In 2021 Asian Conference on Innovation in Technology (ASIANCON). Retrieved March 25, 2024, from https://doi.org/10.1109/asiancon51346.2021.9544689.

Moon, N. N., Sharmin, M. S., Hossain, R. A., Jahan, I., Nur, F. N., & Rahman, M. M. (2021). Deep learning model for detecting and diagnosing plant disease. In *In 2021 International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON)*. https://doi.org/10.1109/smartgencon51891.2021.9645857

Nag, A., Chanda, P. R., & Nandi, S. (2023). Mobile app-based tomato disease identification with fine-tuned convolutional neural networks. *Computers and Electrical Engineering, 112*, Article 108995. https://doi.org/10.1016/j.compeleceng.2023.108995

Nasra, P., & Gupta, S. (2024). Efficient orange disease detection using MobileNetV2: A deep learning approach. *International Conference on IoT Based Control Networks and Intelligent Systems (ICICNIS), 2024*, 900–905. https://doi.org/10.1109/icicnis64247.2024.10823130

Pan, W., Qin, J., Xiang, X., Wu, Y., Tan, Y., & Xiang, L. (2019). A smart mobile diagnosis system for citrus diseases based on densely connected convolutional networks. *IEEE Access, 7*, 87534–87542. https://doi.org/10.1109/access.2019.2924973

Pardede, J., Sitohang, B., Akbar, S., & Khodra, M. L. (2021). Implementation of transfer learning using VGG16 on fruit ripeness detection. *International Journal of Intelligent Systems and Applications, 13*(2), 52–61. https://doi.org/10.5815/ijisa.2021.02.04

Paymode, A. S., Magar, S. P., & Malode, V. B. (2021). Tomato leaf disease detection and classification using convolution neural network. In 2021 International Conference on Emerging Smart Computing and Informatics (ESCI). Retrieved March 25, 2024, from https://doi.org/10.1109/esci50559.2021.9397001.

Saranya, N., Pavithra, L., Kanthimathi, N., Ragavi, B., & Sandhiyadevi, P. (2020, July). Detection of banana leaf and fruit diseases using neural networks. In 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA). Retrieved March 25, 2024, from https://doi.org/10.1109/icirca48905.2020.9183006.

Saragih, R. E., & Emanuel, A. W. R. (2021). Banana ripeness classification based on deep learning using convolutional neural network. In *In 2021 3rd East Indonesia Conference on Computer and Information Technology (EIConCIT)*. https://doi.org/10.1109/eiconcit50028.2021.9431928

Sharath, D. M., Kumar, S. A., Rohan, M. G., Akhilesh, K. V. Suresh, & Prathap, C. (2020, August). Disease detection in plants using convolutional neural network. In 2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT). Retrieved March 25, 2024, from https://doi.org/10.1109/icssit48917.2020.9214159.

Sharif, M., Khan, M. A., Iqbal, Z., Azam, M. F., Lali, M. I. U., & Javed, M. Y. (2018). Detection and classification of citrus diseases in agriculture based on optimized weighted segmentation and feature selection. *Computers and Electronics in Agriculture, 150*, 220–234. https://doi.org/10.1016/j.compag.2018.04.023

Sophia, S., Devi, D., Lakshmi Prabha, K., Keerthana, V., & Kavin, V. (2021). A novel method to detect disease in leaf using deep learning approach. In *In 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*. https://doi.org/10.1109/icaccs51430.2021.9441892

Storey, G., Meng, Q., & Li, B. (2022). Leaf disease segmentation and detection in apple orchards for precise smart spraying in sustainable agriculture. *Sustainability, 14*(3), 1458. https://doi.org/10.3390/su14031458

Sudana, O., Witarsyah, D., Putra, A., & Raharja, S. (2020). Mobile application for identification of coffee fruit maturity using digital image processing. *International Journal on Advanced Science, Engineering and Information Technology, 10*(3), 980–986. https://doi.org/10.18517/ijaseit.10.3.11135

Soyer, M. S., Yilmaz, C., Ozcan, I. M., Cogen, F., & Yildiz, T. C. (2021). LeafLife: Deep learning based plant disease detection application. In *In 2021 13th International Conference on Electrical and Electronics Engineering (ELECO)*. https://doi.org/10.23919/eleco54474.2021.9677785

Tembhurne, J. V., Gajbhiye, S. M., Gannarpwar, V. R., Khandait, H. R., Goydani, P. R., & Diwan, T. (2023). Plant disease detection using deep learning-based mobile application. *Multimedia Tools and Applications, 82*(18), 27365–27390. https://doi.org/10.1007/s11042-023-14541-8

TensorFlow. (n.d.). TensorFlow. Retrieved March 27, 2024, from https://www.tensorflow.org.

USDA APHIS. (2024, January 20). Citrus diseases. Retrieved from https://www.aphis.usda.gov/aphis/ourfo/planthealth/plant-pest-and-disease-programs/pests-a-diseases/citrus/ci-land.

Wang, J., Viciano-Tudela, S., Parra, L., Lacuesta, R., & Lloret, J. (2023). Evaluation of suitability of low-cost gas sensors for monitoring indoor and outdoor urban areas. *IEEE Sensors Journal, 23*(18), 20968–20975. https://doi.org/10.1109/jsen.2023.3301651