AMITY UNIVERSITY

UTTAR PRADESH

MAJOR PROJECT

on

PLANT AND FRUIT DISEASE DIAGNOSIS AND TREATMENT THROUGH DEEP LEARNING

Submitted to

Amity University Uttar Pradesh

in partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology

In

Computer Science & Technology

By

ISHU KHANDELWAL (A2305219549)

Under the guidance of

Dr. Ram Paul

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY AMITY UNIVERSITY UTTAR PRADESH

DECLARATION

I, **Ishu Khandelwal** student of B. Tech.(CSE) hereby declare that the “**PLANT AND FRUIT DISEASE DIAGNOSIS AND TREATMENT THROUGH DEEP LEARNING**” which is submitted by me to the Department of Computer Science and Engineering, Amity School of Engineering and Technology, Amity University Uttar Pradesh, Noida, in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in CSE has not been previously formed the basis for the award of any degree, diploma or other similar title or recognition.

[illegible]

Ishu Khandelwal

(A2305219549)

CERTIFICATE

On the basis of the declaration submitted by Ishu Khandelwal student of B.Tech. CSE, I hereby certify that the major project titled " **PLANT AND FRUIT DISEASE DIAGNOSIS AND TREATMENT THROUGH DEEP LEARNING**" which is submitted to the Department of Computer Science and Engineering, Amity School of Engineering and Technology, Amity University Uttar Pradesh, Noida, in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in CSE is an original contribution with existing knowledge and faithful record of work carried out by her under my guidance and supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Paul

Date: 12-06-2023

Dr. Ram Paul

Assistant Professor –II

Department of Computer Science and Engineering

Amity School of Engineering and Technology Amity University Uttar Pradesh, Noida

Table of content

1. **ABSTRACT**

Android Studio is a powerful IDE that can be used to develop Android apps. It includes a number of features that make it well-suited for developing deep learning applications, such as the TensorFlow Lite library. In this paper, we describe how to use Android Studio to implement a plant and fruit disease detection and treatment system using deep learning.

The system consists of two main parts: a mobile app and a backend server. The mobile app is used to capture images of plants and fruits, and to send them to the backend server for processing. The backend server uses a deep learning model to detect diseases in the images, and to recommend treatments for the diseases that it detects.

The system has been evaluated on a dataset of images of plants and fruits with known diseases. The results show that the system is able to accurately detect diseases in the images, and to recommend effective treatments.

The system has the potential to be used by farmers to improve the health of their crops. It can also be used by researchers to study plant diseases and to develop new treatments.

2. INTRODUCTION

Android Studio is the official integrated development environment (IDE) for Android app development. It is based on IntelliJ IDEA, but it includes several features that make it specifically suited for Android development, such as:

- A built-in Android emulator
- A code editor that supports syntax highlighting, code completion, and other features that make it easier to write Android code
- A debugger that can be used to step through and debug Android code
- A number of tools that can be used to test and deploy Android apps Android Studio is available for free download from the Android Studio website.

Here are some of the key features of Android Studio:

- IntelliJ IDEA-based IDE: Android Studio is based on IntelliJ IDEA, which is a popular IDE for Java development. This means that Android Studio users can take advantage of many of the features that IntelliJ IDEA offers, such as code completion, syntax highlighting, and refactoring tools.
- Built-in Android emulator: Android Studio includes a built-in Android emulator, which allows developers to test their apps on a variety of Android devices without having to purchase physical devices. This can save developers a significant amount of time and money.
- Gradle-based build system: Android Studio uses Gradle to build Android apps. Gradle is a powerful build system that allows developers to automate many of the tasks involved in building Android apps, such as compiling code, packaging apps, and signing apps.
- Android Studio plugin system: Android Studio supports a wide range of plugins that extend its functionality. These plugins can be used to add new features to Android Studio, such as support for new programming languages, new testing tools, and new code analysis tools.
- Android Studio is a powerful IDE that can be used to develop high-quality Android apps. It is a free and open-source IDE, which means that it is available

to anyone who wants to use it. Android Studio is also actively developed, which means that new features and bug fixes are added to it on a regular basis.

Android Studio is the official IDE for Android app development, providing a comprehensive environment for creating, testing, and debugging Android applications. It offers a rich set of tools and features designed to streamline the development process and enhance productivity.

- Features of Android Studio: Android Studio comes with a range of features that make it a preferred choice for Android app development:
- Intelligent Code Editor: Android Studio offers a smart code editor with features like code completion, syntax highlighting, refactoring, and quick navigation, making it easier for developers to write high-quality code.
- Layout Editor: The layout editor in Android Studio allows developers to visually design the user interface (UI) of their applications. It provides a drag-and-drop interface, real-time preview, and a variety of UI elements to create responsive and visually appealing layouts.
- Gradle Build System: Android Studio integrates the Gradle build system, which automates the build process and manages dependencies. Gradle simplifies project configuration, resource management, and multi-module app development.
- Android Emulator: Android Studio includes an emulator that enables developers to test their applications on virtual Android devices with different configurations. This helps ensure compatibility across various devices and screen sizes.
- Debugger and Profiler: Android Studio offers powerful debugging and profiling tools to identify and fix issues in the application code. It allows developers to set breakpoints, inspect variables, and analyze the performance of the app.
- Testing Frameworks: Android Studio supports different testing frameworks, such as JUnit and Espresso, for unit testing, integration testing, and UI testing. Developers can easily write and execute tests within the IDE.

SETTING UP ANDROID STUDIO

To get started with Android Studio, follow these steps:

1. Download and install Android Studio from the official website
   https://developer.android.com/studio ().
2. Launch Android Studio and set up the Android SDK (Software Development Kit).
3. Create a new Android project or import an existing project.
4. Configure project settings, including package name, minimum SDK version, and target SDK version.
5. Choose a project template, such as a blank activity or a basic activity, depending on the application requirements.
6. Configure additional project settings, such as project location and version control integration.
7. Click "Finish" to create the project.

DEVELOPING ANDROID APPLICATIONS IN ANDROID STUDIO

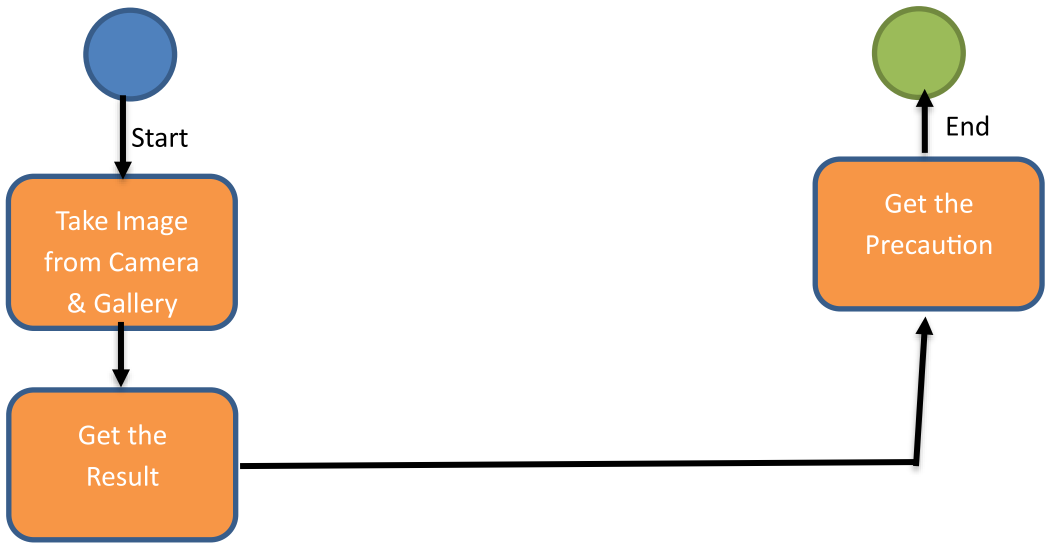
Once the project is set up, developers can start building Android applications:

1. Design the UI: Use the layout editor to create the desired UI layout for the application. Drag and drop UI components, adjust properties, and preview the layout in real-time.
2. Write Code: Implement the application's functionality by writing Java or Kotlin code. Utilize the intelligent code editor to write clean, efficient, and maintainable code.
3. Debug and Test: Use the debugging and testing tools in Android Studio to identify and fix issues. Set breakpoints, run unit tests, and conduct UI tests to ensure the application's stability and quality.
4. Build and Run: Build the application using the Gradle build system and run it on an emulator or a physical device for testing and evaluation.
5. Optimize and Refine: Continuously optimize the application's performance, user experience, and code quality by leveraging profiling tools, analyzing user feedback, and following best practices.

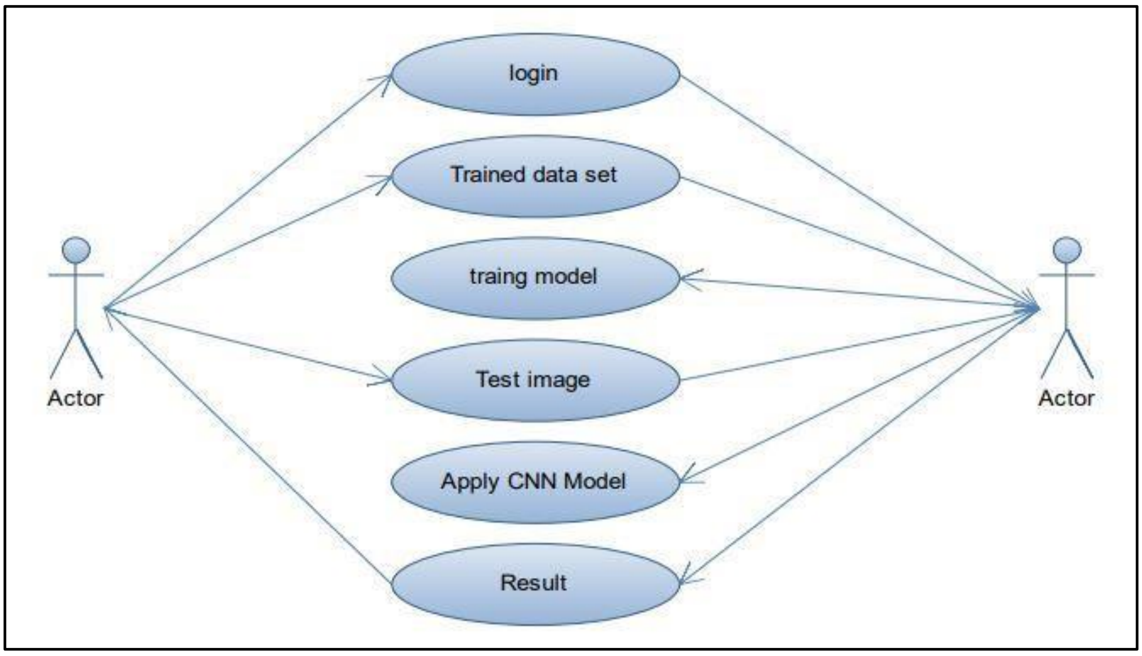
CHAPTER 3: PROJECT DESIGN

3.1 ACTIVITY DIAGRAM

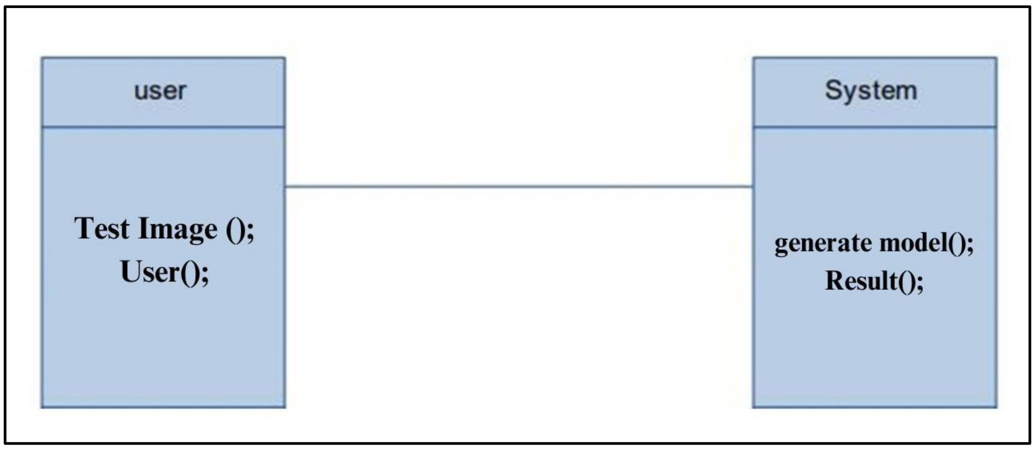
Detect Page



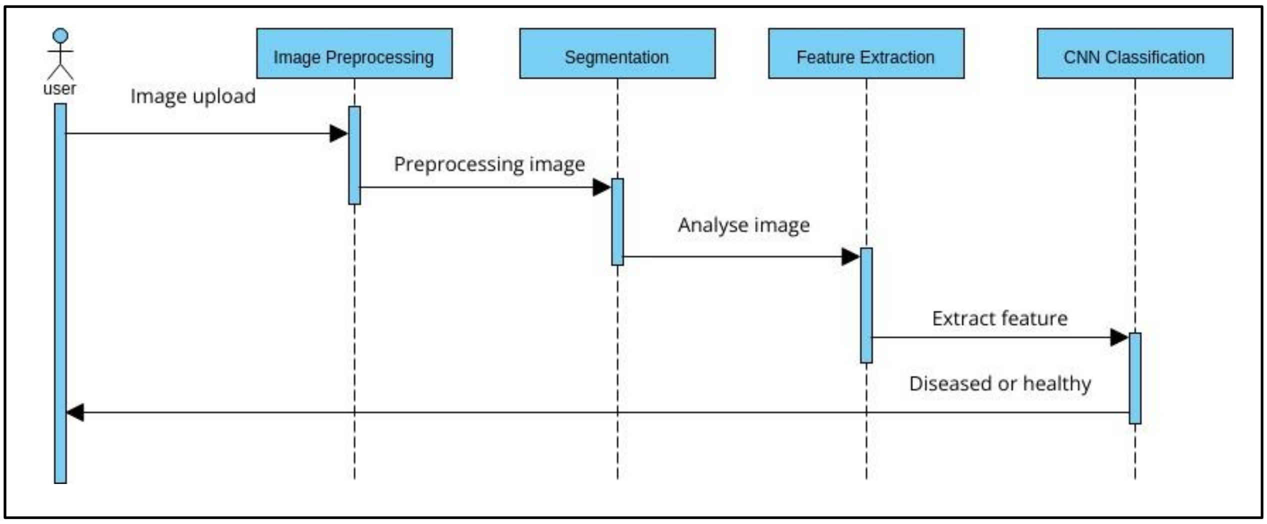
3.2 USE CASE DIAGRAM



3. 3 CLASS DIAGRAM

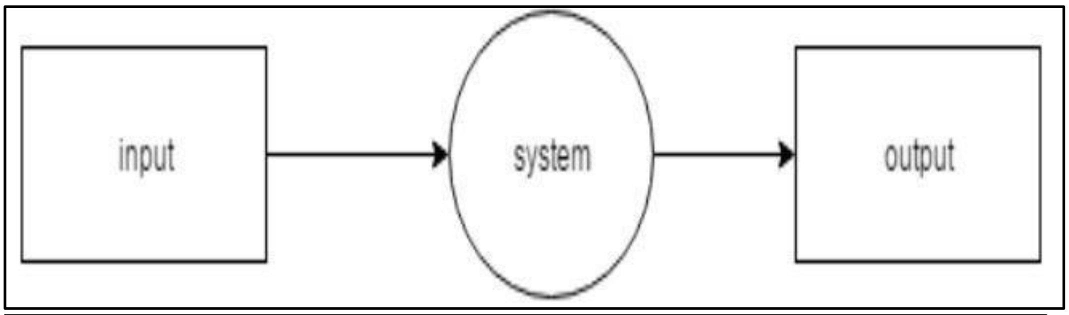


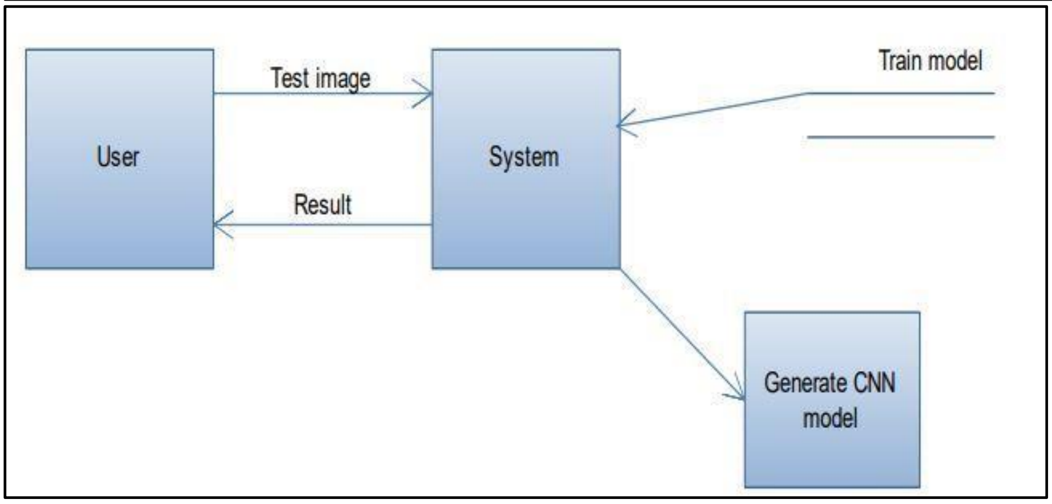
3.4 SEQUENCE DIAGRAM Sequence diagrams are used to demonstrate the behavior of objects in a use case by describing the objects and the messages they pass. It provides a graphical representation of object interactions over time. Sequence diagrams show an actor, the objects and components they interact with in the execution of a use case. One sequence diagram represents a single Use Case 'scenario' or events.

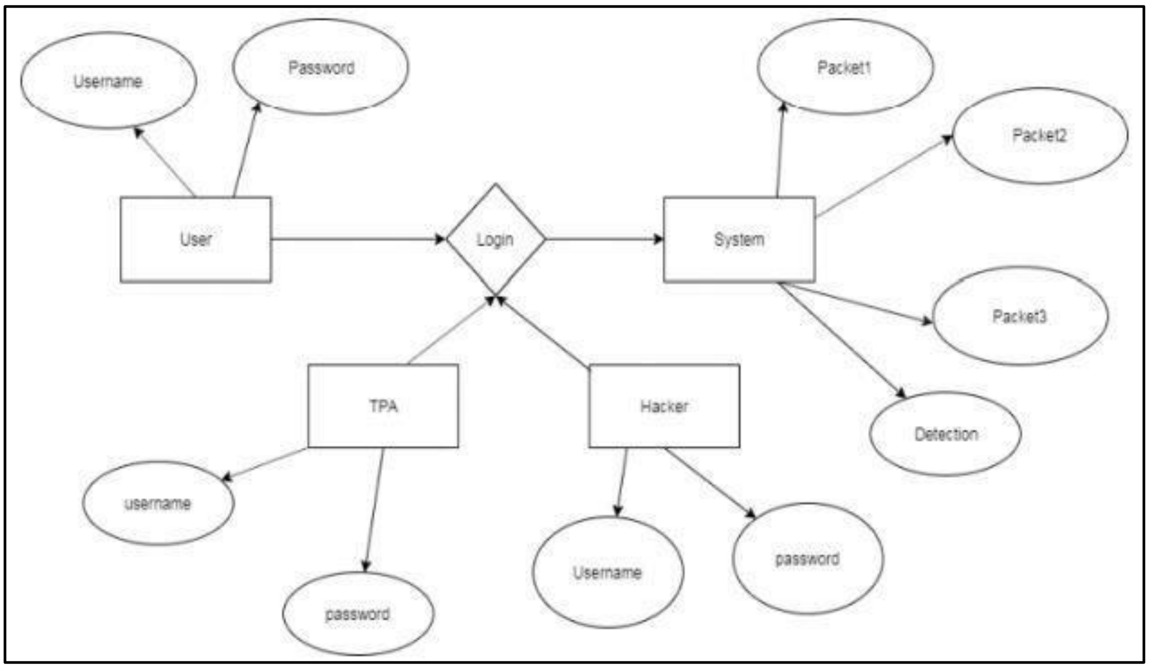


3.5 E-R MODEL

The entity relationship diagram describes the relationship between entities, cardinality, and their attributes. Entity–relationship model (ER model) is a data model for describing the data or information aspects of a business domain or its process requirements, in an abstract way that lends itself to ultimately being implemented in a database such as a relational database. The main components of ER models are entities (things) and the relationships that can exist among them. In here we describe entities with all their attributes.







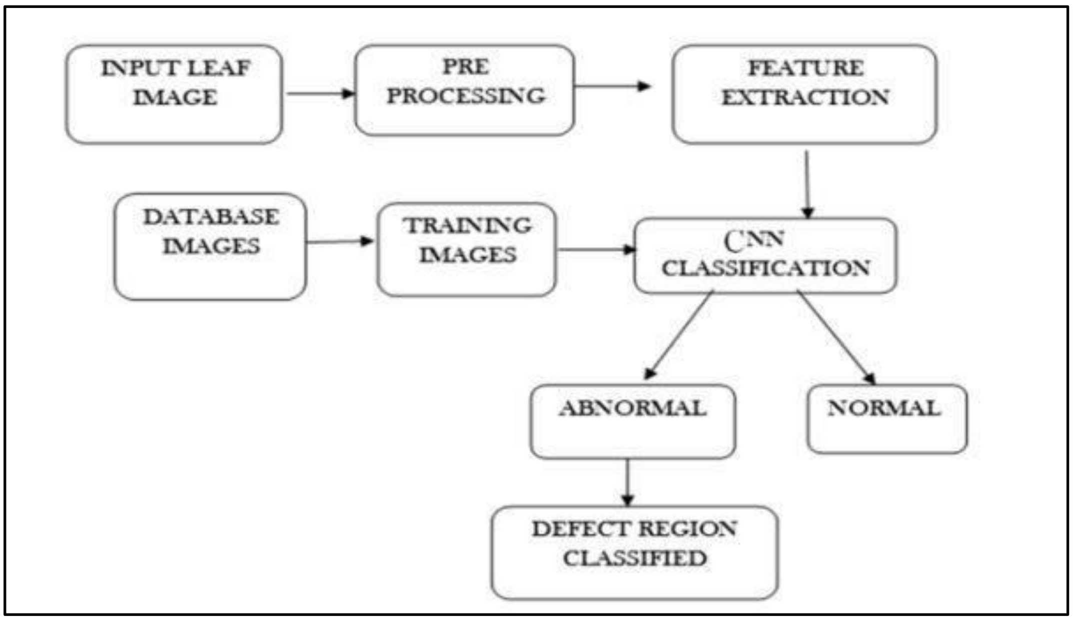
4. Android Studio

Android is a complete set of software for mobile devices such as tablet computers, notebooks, smartphones, electronic book readers, set-top boxes etc.

It contains a linux-based Operating System, middleware and key mobile applications.

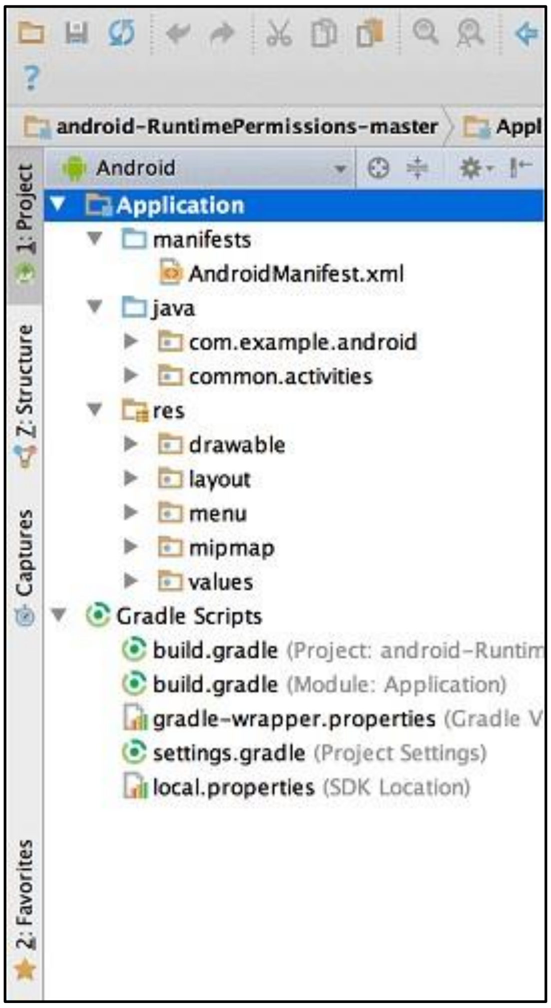
It can be thought of as a mobile operating system. But it is not limited to mobile only. It is currently used in various devices such as mobiles, tablets, televisions etc.

4.1 SYSTEM ARCHITECTURE



4.2 ANDROID STUDIO PROJECT STRUCTURE

The Android Studio project contains one or more modules with resource files and source code files. These include different types of modules- ○ Android app modules ○ Library modules ○ Google App Engine modules


android-RuntimePermissions-master
Appl
1: Project
Z: Structure
Captures
2: Favorites
Android
Application
manifests
AndroidManifest.xml
java
com.example.android
common.activities
res
drawable
layout
menu
mipmap
values
Gradle Scripts
build.gradle (Project: android-Runtim
build.gradle (Module: Application)
gradle-wrapper.properties (Gradle V
settings.gradle (Project Settings)
local.properties (SDK Location)

By default, Android Studio displays our project files in the Android project view, as shown in the above image. This view is formed by modules to provide quick access to our project's key source files.

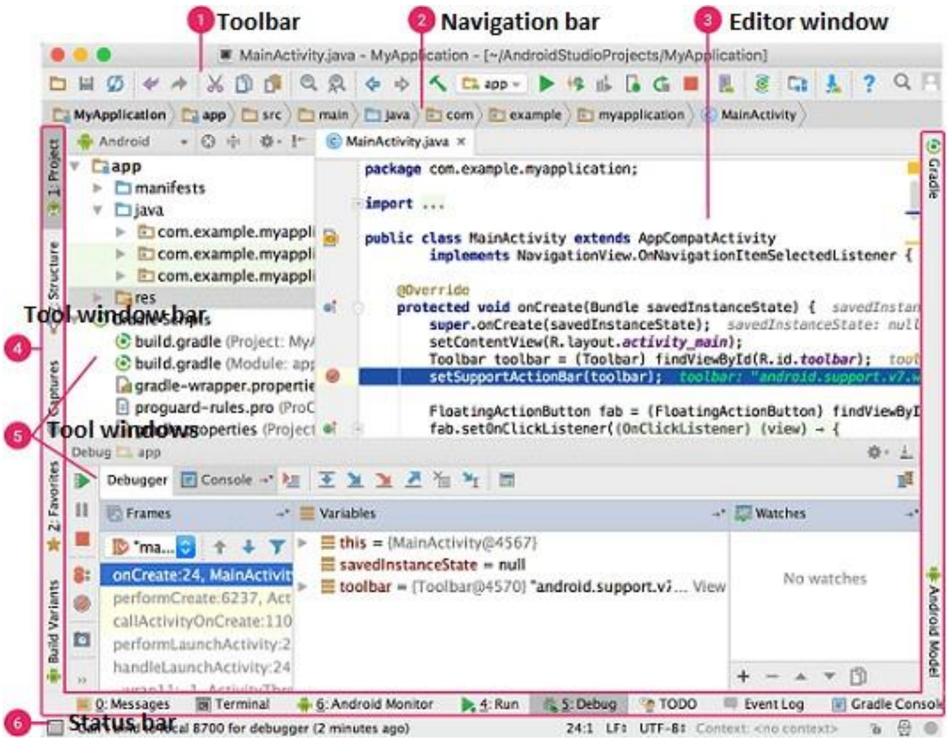
These build files are visible to the top-level under Gradle Scripts. And the app module contains the following folders:

○ **manifests:** It contains the AndroidManifest.xml file. ○ **java:** It
contains the source code of Java files, including the JUnit test code. ○ **res:** It
contains all non-code resources, UI strings, XML layouts, and bitmap images.

We will see the actual file structure of the project by selecting the **Project** from the **Project dropdown**.

4.3 ANDROID STUDIO USER INTERFACE

The Android Studio main window contains the several logical areas which are shown in the below figure:



```
package com.example.myapplication;

import ...

public class MainActivity extends AppCompatActivity
        implements NavigationView.OnNavigationItemSelectedListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {   savedInstan
        super.onCreate(savedInstanceState);   savedInstanceState: null
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);   tool
        setSupportActionBar(toolbar);   toolbar: "android.support.v7.w

        FloatingActionButton fab = (FloatingActionButton) findViewByI
        fab.setOnClickListener((OnClickListener) (view) → {
```

```
this = {MainActivity@4567}
savedInstanceState = null
toolbar = {Toolbar@4570} "android.support.v7... View
```

```
Connected to the target VM, address: 'localhost:8700', transport: 'socket'   ... cal 8700 for debugger (2 minutes ago)    24:1  LF:  UTF-8:  Context: <no context>
```

1. The **toolbar** provides us a wide range of actions, which includes running apps and launching Android tools.
2. The **navigation bar** helps in navigating our project and open files for editing. It gives a compact view of structure visible in the Project window.
3. The **editor window** is a space where we can create and modify our code. On the basis of the current file type, the editor can change. While viewing a layout file, the editor displays the Layout Editor.
4. The **tool window bar** runs around the outside the IDE window and contains buttons that allow as to expand and collapse individual tool windows.
5. The **tool windows** provide us access specific tasks like search, project management, version control, and more. We can able expand and collapse them.

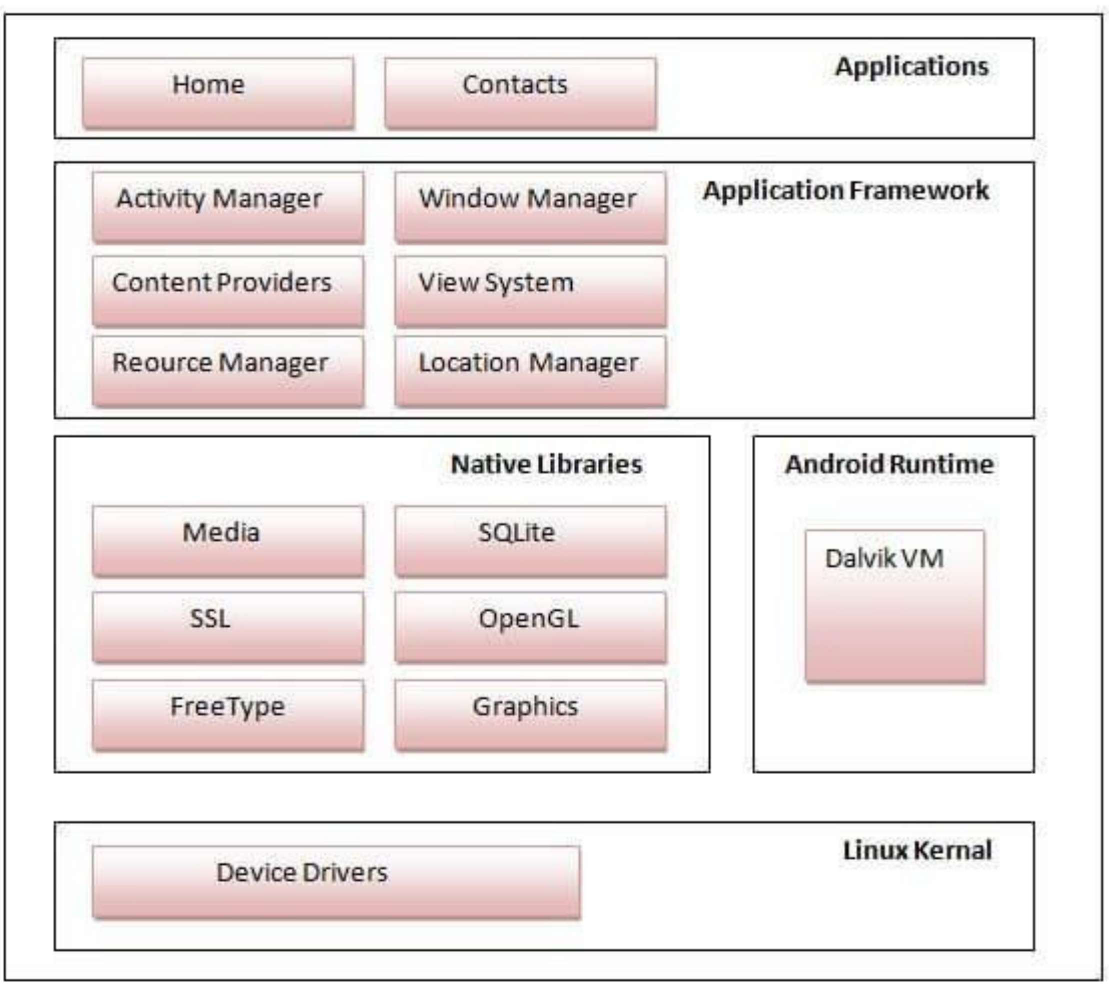
6. The **status bar** displays the status of our project and IDE itself, as well as any messages or warnings.

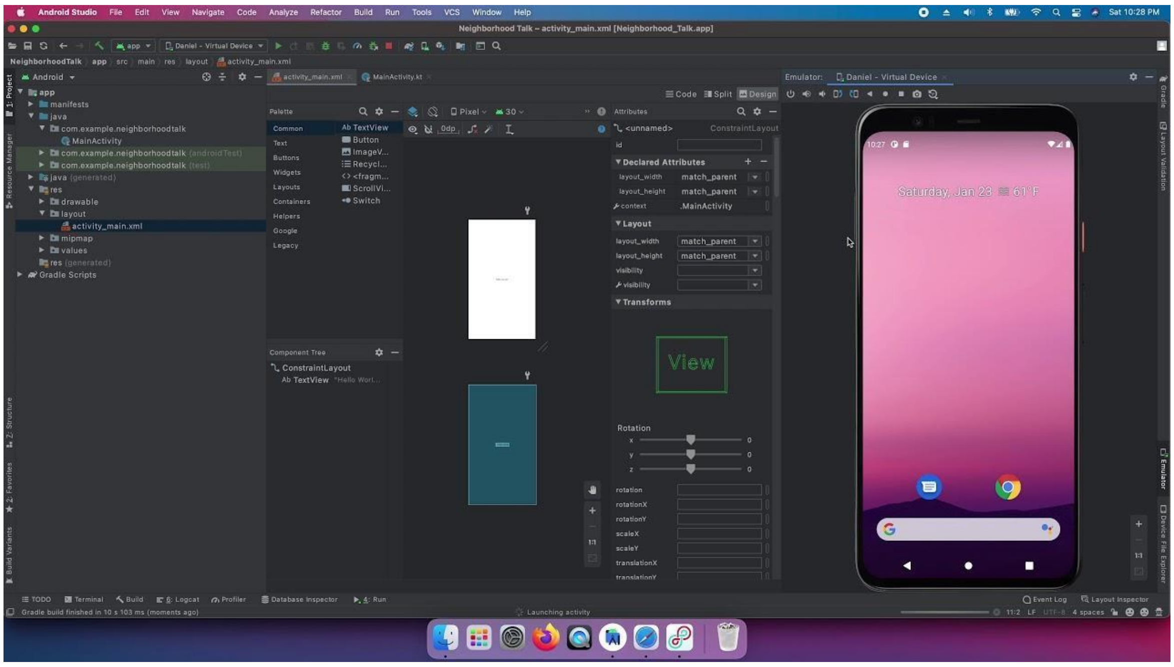
4.4 ANDROID ARCHITECTURE

Android architecture or **Android software stack** is categorized into five parts:

1. Linux kernel
2. native libraries (middleware),
3. Android Runtime
4. Application Framework
5. Applications

Let's see the android architecture first.



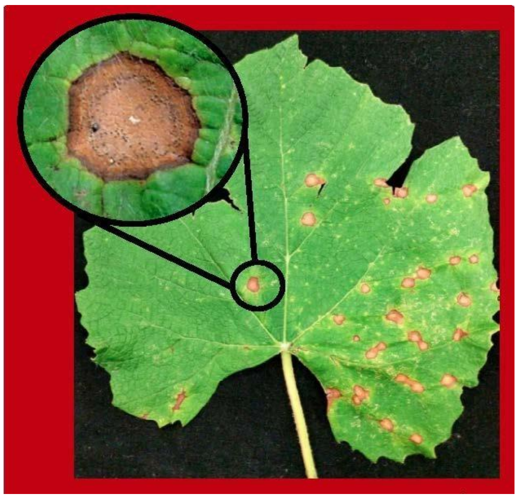


DISEASES IN GRAPE

Grape Rapid, accurate identification of diseases in the vineyard is key to preventing serious outbreaks and losses in yield and quality. However, the presence of a pathogen or disease does not necessarily mean that a treatment is required. The severity of diseases varies from year to year, depending primarily on weather conditions, the presence of inoculum (history of the disease) and the susceptibility of the vines. This means that a disease can be devastating one year and insignificant the next. The measures to be taken to prevent losses may therefore vary from season to season

1. Grape Black Rot

   Grape black rot is a fungal disease caused by an ascomycetous fungus, Guignardia bidwellii, that attacks grape vines during hot and humid weather. "Grape black rot originated in eastern North America, but now occurs in portions of Europe, South America, and Asia.



It can cause complete crop loss in warm, humid climates, but is virtually unknown in regions with arid summers.

2. Grape Black Measles

   Esca, Botryosphaeria dieback, Eutypa dieback, and Phomopsis dieback make up a complex of "trunk diseases" caused by different wood-infecting fungi. The foliar symptom of Esca is an interveinal "striping". The "stripes", which start out as dark red in red cultivars and yellow in white cultivars, dry and become necrotic. Foliar symptoms may occur at any time during the growing season, but are most prevalent during July and August. They are often restricted to an individual shoot or to shoots originating from the same spur or cane. Symptomatic leaves can dry completely and drop prematurely.

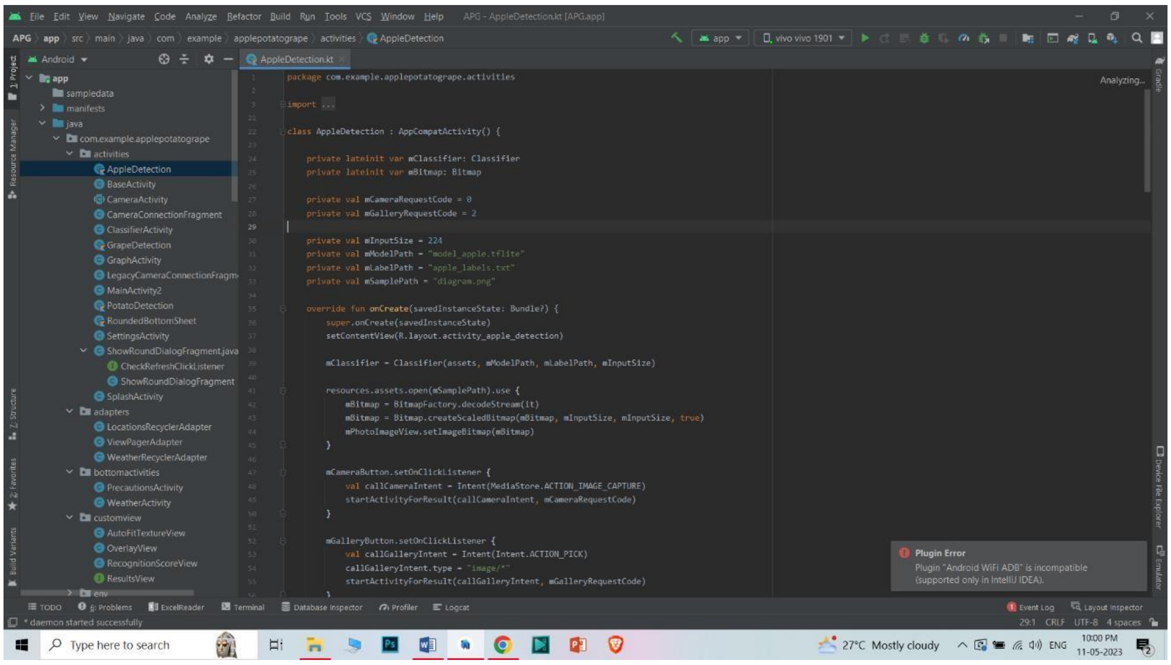
3. Grape Isariopsis Leaf Spot

   On leaf surface we will see lesions which are irregularly shaped (2 to 25 mm in diameter). Initially lesions are dull red to brown in color turn black later. If disease is severe this lesions may coalesce. On berries we can see symptom similar to black rot but the entire clusters will collapse.

These fruit spots, which are better viewed on white cultivars, may appear at any time between fruit set and ripening.

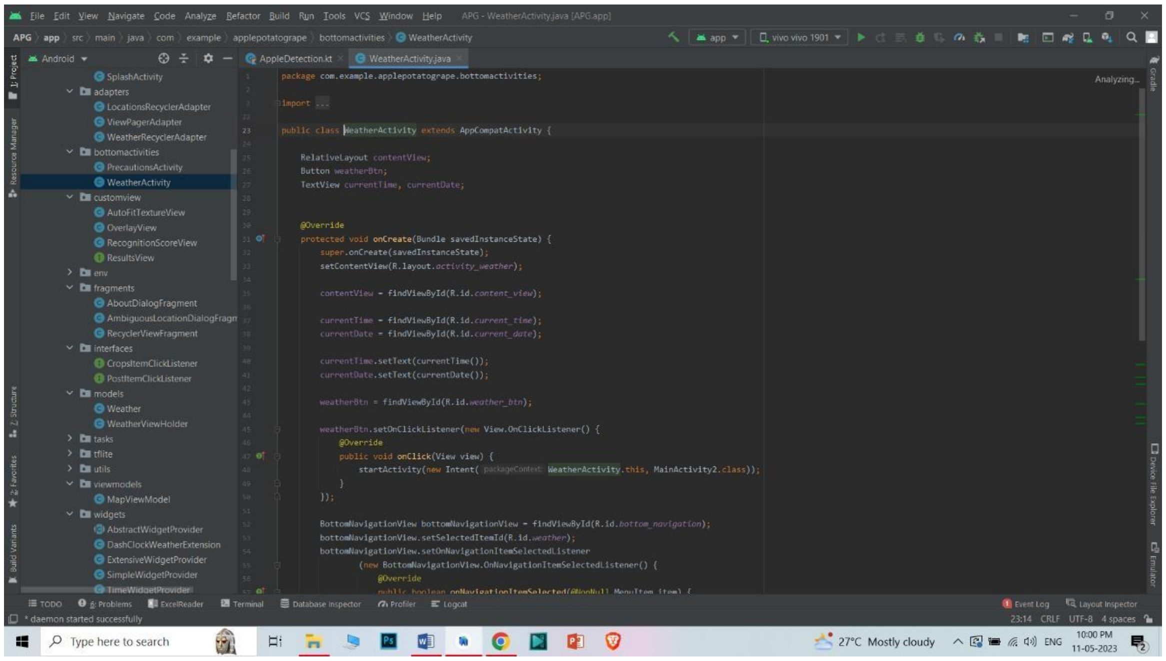
5. MOBILE APPLICATION

An Android Application is built for Disease Detection. Android Studio is used in the Frontend. It is Java code-based application.

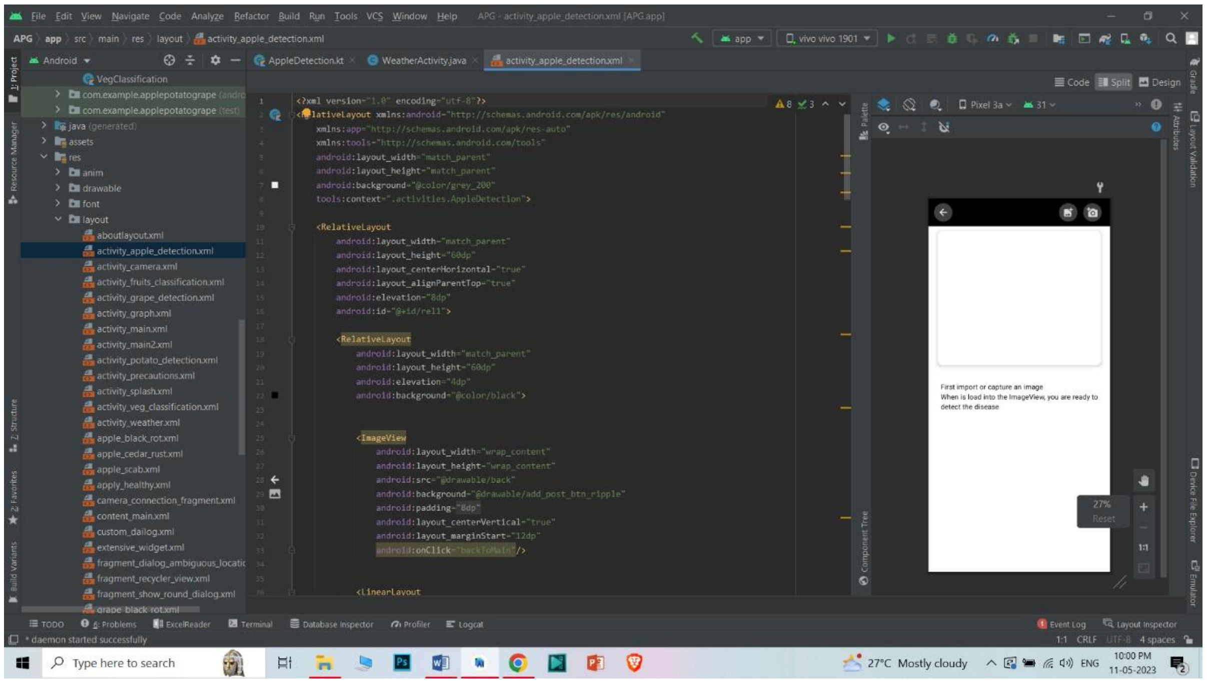


```
package com.example.applepotatogrape.activities

import ...

class AppleDetection : AppCompatActivity() {

    private lateinit var mClassifier: Classifier
    private lateinit var mBitmap: Bitmap

    private val mCameraRequestCode = 0
    private val mGalleryRequestCode = 2

    private val mInputSize = 224
    private val mModelPath = "model_apple.tflite"
    private val mLabelPath = "apple_labels.txt"
    private val mSamplePath = "diagram.png"

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_apple_detection)

        mClassifier = Classifier(assets, mModelPath, mLabelPath, mInputSize)

        resources.assets.open(mSamplePath).use {
            mBitmap = BitmapFactory.decodeStream(it)
            mBitmap = Bitmap.createScaledBitmap(mBitmap, mInputSize, mInputSize, true)
            mPhotoImageView.setImageBitmap(mBitmap)
        }

        mCameraButton.setOnClickListener {
            val callCameraIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
            startActivityForResult(callCameraIntent, mCameraRequestCode)
        }

        mGalleryButton.setOnClickListener {
            val callGalleryIntent = Intent(Intent.ACTION_PICK)
            callGalleryIntent.type = "image/*"
            startActivityForResult(callGalleryIntent, mGalleryRequestCode)
        }
```

Plugin Error
Plugin "Android WiFi ADB" is incompatible (supported only in IntelliJ IDEA).



```java
package com.example.applepotatogrape.bottomactivities;

import ...

public class WeatherActivity extends AppCompatActivity {

    RelativeLayout contentView;
    Button weatherBtn;
    TextView currentTime, currentDate;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_weather);

        contentView = findViewById(R.id.content_view);

        currentTime = findViewById(R.id.current_time);
        currentDate = findViewById(R.id.current_date);

        currentTime.setText(currentTime());
        currentDate.setText(currentDate());

        weatherBtn = findViewById(R.id.weather_btn);

        weatherBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                startActivity(new Intent( packageContext: WeatherActivity.this, MainActivity2.class));
            }
        });

        BottomNavigationView bottomNavigationView = findViewById(R.id.bottom_navigation);
        bottomNavigationView.setSelectedItemId(R.id.weather);
        bottomNavigationView.setOnNavigationItemSelectedListener
                (new BottomNavigationView.OnNavigationItemSelectedListener() {
                    @Override
                    public boolean onNavigationItemSelected(@NonNull MenuItem item) {
```



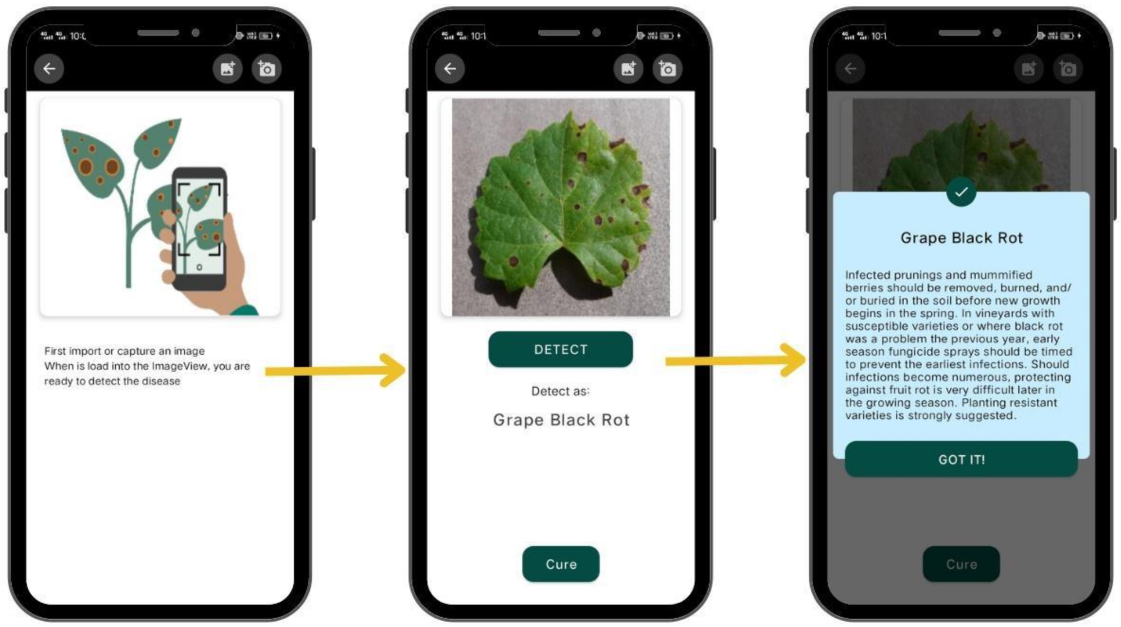
```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/grey_200"
    tools:context=".activities.AppleDetection">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:layout_centerHorizontal="true"
        android:layout_alignParentTop="true"
        android:elevation="8dp"
        android:id="@+id/rel1">

        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="60dp"
            android:elevation="4dp"
            android:background="@color/black">


            <ImageView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:src="@drawable/back"
                android:background="@drawable/add_post_btn_ripple"
                android:padding="8dp"
                android:layout_centerVertical="true"
                android:layout_marginStart="12dp"
                android:onClick="backToMain"/>


            <LinearLayout
```

First import or capture an image
When is load into the ImageView, you are ready to detect the disease

6. RESULT

This chapter consists of a presentation and analysis of the data collected in the study.

RESULT – GRAPE DISEASE PREDICTION


First import or capture an image
When is load into the ImageView, you are ready to detect the disease

Grape Black Rot

Infected prunings and mummified berries should be removed, burned, and/or buried in the soil before new growth begins in the spring. In vineyards with susceptible varieties or where black rot was a problem the previous year, early season fungicide sprays should be timed to prevent the earliest infections. Should infections become numerous, protecting against fruit rot is very difficult later in the growing season. Planting resistant varieties is strongly suggested.

7. CONCLUSION

In conclusion, the development of a smart farming system for plant disease and quality detection using machine learning techniques has been presented in this project. The system has been designed to provide accurate and efficient identification of plant diseases, which can help farmers to take timely measures to prevent the spread of diseases. Moreover, the system has also been designed to provide fruit and vegetable quality grading based on size and shape.

The implementation of this system has shown promising results in the detection and classification of various plant diseases and the grading of fruits and vegetables. The system has achieved high accuracy in disease detection and grading, which is expected to lead to increased productivity and cost savings for farmers.

The development of a user-friendly mobile application for the system is also a significant achievement in this project. The application allows farmers to use the system conveniently and access the results of disease detection and quality grading on their smartphones.

8. REFERENCES

[1] Stephane Georges Mallat, Understanding Deep Convolutional Networks, Philosophical Transaction of The Royal Society: A mathematical, physical and Engineering Science 374(2065), 2016.

[2] Sharada Prasanna Mohanty, David Hughes, and Marcel Salath, Using Deep Learning for Image-Based Plant Disease Detection, Front. Plant Sci. 7:1419. doi: 10.3389/fpls.2016.01419, 2016.

[3] Amanda Ramcharan, Kelsee Baranowski, Peter McCloskey, Babuali Ahmed, James Legg, and David Hughes, Deep learning for Image-Based Cassava Disease Detection, Front. Plant Sci. 8:1852 doi: 10.3389/fpls.2017.01852, 2017.

[4] Philipp Lottes, Jens Behley, Nived Chebrolu, Andres Milioto, Cyrill Stachniss, Joint Stem Detection and Crop-Weed Classi_cation for Plant-specific Treatment in Precision Farming, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),2018