Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Spring, Year: 2023), B.Sc. in CSE (Day)

Image Detection Android App

Course Title: Machine Learning Lab
Course Code: CSE 412
Section: 201D5

Students Details

Name	ID
Joy Munshi	201002143
Md. Jahid Hassan	201002463

Submission Date: 14-06-2023
Course Teacher's Name: MS. SADIA AFROZE

[For teachers use only: [Don't write anything inside this box]

Lab Project Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

An image detection app is a mobile application that is able to analyze images and provide information about their contents. These apps use machine learning algorithms [1] to identify objects, people, text, and other subjects within images. Image detection apps can be used for a wide range of purposes, including object recognition, facial recognition, text recognition, and more. To build an image detection app, developers will need to use machine learning techniques to train a model that is able to recognize the objects or features that they want to detect. Once the model is trained, it can be integrated into the app to allow it to make predictions about the contents of images. Image detection apps can be built for various platforms, including iOS and Android, and can be developed using a variety of programming languages and tools.

1.1 Overview

Here is an overview of your image detection Android app:

1. The purpose of the app is to provide an educational tool for kids to learn about different objects by detecting and identifying them in images.
2. The target audience is children [2].
3. The app will allow users to select or take an image, and then use a machine-learning model to detect and identify the objects in the image. Users will be able to initiate the object detection process by clicking a "predict" button.
4. The technology stack for the project will include Android Studio and the Java programming language, as well as a machine learning framework or library for training the object detection model.
5. The project timeline is planned for the duration of your university semester.
6. As a university student project, there is no budget allocated for the app.

7. Potential risks and challenges include obtaining a high-quality machine learning model for object detection and ensuring that the object detection functionality of the app is accurate and reliable. These challenges will need to be addressed through careful experimentation and testing.

1.2 Motivation

There are many potential motivations for building an image detection app, depending on the specific goals and needs of the developers and users. Some possible motivations for building an image detection app might include:

- To create a useful and engaging tool for a specific purpose, such as education or entertainment. For example, an image detection app for kids could be used to teach them about different objects, animals, or other subjects.
- To solve a specific problem or meet a specific need. For example, an image detection app might be used to help visually impaired people identify objects or navigate their environment.
- To showcase a new technology or demonstrate the capabilities of machine learning. Image detection apps can be used to demonstrate the capabilities of machine learning algorithms and show how they can be applied to real-world problems.
- To generate revenue through advertising or in-app purchases. Image detection apps can be monetized by displaying advertisements or offering premium features that can be purchased within the app.

Overall, the motivation for building an image detection app will depend on the specific goals and needs of the developers and the intended audience.

1.3 Problem Definition

1.3.1 Problem Statement

The problem statement for an image detection app will depend on the specific goals and needs of the developers and users. Some possible problem statements for an image detection app might include:

- "How can we create an educational tool that helps kids learn about different objects by detecting and identifying them in images?"
- "How can we use machine learning to build an app that helps visually impaired people identify objects and navigate their environment?"
- "How can we use image detection to improve the accuracy of OCR (optical character recognition) for text recognition in images?"

- "How can we create an image detection app that is able to accurately identify and classify a wide range of objects in real-time?"

In each of these cases, the problem statement defines the specific challenge that the image detection app is intended to address. By defining the problem clearly, developers can better understand the requirements and constraints of the project, and can focus their efforts on creating a solution that meets the needs of the users.

1.3.2 Complex Engineering Problem

One example of a complex engineering problem that could be addressed using image detection and machine learning is the task of automating the inspection of industrial equipment. In this scenario, the problem might be stated as follows:

Table 1.1: Summary of the attributes touched by the mentioned projects

Name of the P Attributess	**Explain how to address**
Speed: Range of conflicting requirements	TThe system must be able to process images and make predictions quickly, in order to keep up with the pace of the production line.
Scalability: Depth of analysis required	The system must be able to handle a large volume of images and work with a wide range of equipment types and configurations.
Robustness: Depth of analysis required	The system must be able to handle variations in lighting and other environmental conditions, and must be able to operate reliably over time.
Cost: Depth of analysis required	The system must be cost-effective to deploy and maintain.

Solving this problem would require a combination of advanced image detection and machine learning algorithms, as well as robust hardware and software infrastructure to support the system. It would also require careful testing and validation to ensure that the system is accurate, reliable, and able to meet the needs of the users.

1.4 Design Goals/Objectives

Objectives are specific goals or targets that an app is intended to achieve. In the context of an image detection app, some possible objectives might include:

- To accurately detect and classify a wide range of objects in images, with a low rate of false positives and false negatives.
- To provide fast and responsive image processing, so that users do not have to wait long for results.
- To offer an easy-to-use and intuitive user interface, so that users can easily interact with the app.
- To be able to handle a large volume of images and work with a wide range of objects and image types.
- To be robust and reliable, able to handle variations in lighting and other environmental conditions and operate over an extended period of time.
- To be cost-effective to develop and maintain.

By defining clear objectives, developers can ensure that they are creating an app that meets the needs of the users and addresses the problem that it was designed to solve.

1.5 Application

Here are some examples of applications for image detection apps that could be designed specifically for kids:

- Educational game: An image detection app could be used to create an educational game that helps kids learn about different objects, animals, or other subjects by detecting and identifying them in images. The game could include a variety of interactive activities, such as matching objects to their names or categories, or solving puzzles that involve identifying objects in images.
- Virtual pet: An image detection app could be used to create a virtual pet that kids can care for and interact with by taking pictures of different objects and foods. The app could use image detection to recognize the objects in the images and respond appropriately, such as by feeding the virtual pet or providing it with toys to play with.
- Story creator: An image detection app could be used to create a story creation tool that kids can use to build their own stories using images and text. The app could use image detection to recognize and classify objects in the images, and then suggest appropriate words or phrases that kids can use to describe the objects and create a story.

- Object identification tool: An image detection app could be used to create a tool that helps kids learn about different objects and their names by taking pictures of them. The app could use image detection to recognize the objects in the images and provide the correct names or labels for them.

These are just a few examples of the many possible applications for image detection apps that could be designed specifically for kids. The specific use case will depend on the goals and needs of the developers and users.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

The design, development, and implementation of an image detection app involves a number of steps and activities. The process typically begins by defining the problem that the app is intended to solve and the specific goals and objectives that it should achieve. This helps to guide the design and development process and ensure that the app meets the needs of the users. The next step is to design the user interface (UI) for the app, which involves creating the screens and layout, and adding UI elements such as buttons, text fields, and images. The UI should be designed to be easy to use and intuitive, so that users can easily interact with the app. After the UI is designed, the next step is to train a machine learning model that is able to detect and classify the objects or features that you want the app to recognize. This may involve using a machine learning framework or library, such as TensorFlow, to build and train the model. Once the model is trained, it can be integrated into the app using the appropriate machine learning framework or library. This may involve using the TensorFlow Lite Interpreter to run inference on the device, or using a server-based solution to make predictions. Finally, the app can be tested and debugged to ensure that it is working correctly and meeting the requirements of the users.

2.2 Project Details

Our university smart network system project aims to create a secure and efficient network for a university that can connect all of the various devices used by students, faculty, and staff. This system will use a variety of technologies, including Telnet, Remote SSH, firewalls, DHCP, VLANs, and smart devices, to achieve its goals.

2.2.1 Key Components

There are several key components that are typically involved in the design, development, and implementation of an image detection app. These components may include:

1. Machine learning model: A machine learning model is a mathematical representation of a system that has been trained to recognize patterns and make predictions based on data. In the context of an image detection app, the model is responsible for analyzing images and identifying the objects or features that the app is designed to detect.

2. User interface: The user interface (UI) is the part of the app that the user interacts with. It includes the screens, layout, and UI elements such as buttons, text fields, and images that the user uses to interact with the app.

3. Data storage: Image detection apps often require the ability to store and retrieve data, such as images, machine learning models, and user preferences. This may involve using a local database on the device or a cloud-based storage solution.

4. Network communication: In some cases, image detection apps may need to communicate with a server or other remote resources in order to retrieve data or make predictions. This may involve using network protocols such as HTTP or WebSockets to transfer data.

5. Image processing: Image detection apps often involve the processing of images in order to extract information from them. This may involve techniques such as filtering, resizing, or cropping images to prepare them for analysis by the machine learning model.

Overall, these are some of the key components that are typically involved in the design, development, and implementation of an image detection app.

2.3 Implementation

2.3.1 Planning

The workflow

The workflow for an image detection app typically involves the following steps:

1. The user selects or takes an image. This may involve using the device's camera or selecting an image from the device's storage.

2. The app processes the image to prepare it for analysis by the machine learning model. This may involve resizing, cropping, or filtering the image to extract relevant features.

3. The app runs the machine learning model on the image to make predictions about the objects or features that it contains.

4. The app displays the results of the prediction to the user, either as text or as an overlay on the image itself.

5. The user may have the option to save the image and the prediction results, or to share them with others.

This is a general overview of the workflow for an image detection app. The specific steps and features will depend on the needs and goals of the app, and may vary from one app to another.

2.3.2 Componets Required

Tools and libraries

There are a variety of tools and libraries that can be used to build an image detection app. Some options that you might consider include:

- Machine learning frameworks: Machine learning frameworks such as TensorFlow and PyTorch provide a set of tools and libraries for building and training machine learning models. These frameworks can be used to build models for object detection, facial recognition, and other image classification tasks.

- Image processing libraries: Libraries such as OpenCV and Pillow provide a range of functions for processing and manipulating images. These libraries can be used to resize, crop, and filter images to prepare them for analysis by a machine learning model.

- Mobile development platforms: Platforms such as Android Studio and Xcode provide tools and libraries for building mobile apps for Android and iOS devices, respectively. These platforms include tools for designing the UI, integrating machine learning models, and testing and debugging the app.

- Data storage solutions: Solutions such as SQLite and Realm provide databases that can be used to store and retrieve data within a mobile app. These solutions can be used to store images, machine learning models, and other data that is needed by the app.

Overall, there are many tools and libraries that can be used to build an image detection app, depending on the specific needs and goals of the project.

2.4 Implementation Details

Design UI/UX

The main XML layout for an image detection app will typically include the layout and UI elements that are used to display the app's content and allow the user to interact with it. This might include elements such as buttons, text fields, images, and other UI controls. Here is an example of a simple main XML layout for an image detection app:

Listing 2.1: activity$_main.xml$*label*

```
<?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"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:text="Image Classify"
        android:id="@+id/title"
        android:textSize="30dp"
        android:textAlignment="center"
        android:layout_marginTop="20dp"

        />
    <ImageView
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:id="@+id/imageView"
        android:layout_below="@id/title"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Select Image"
        android:layout_below="@id/imageView"
        android:id="@+id/selectBtn"
        android:layout_marginTop="20dp"
        android:layout_centerHorizontal="true"/>
    <Button
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="capture"
        android:layout_below="@id/selectBtn"
        android:id="@+id/captureBtn"
        android:layout_marginTop="20dp"
        android:layout_centerHorizontal="true"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="predict"
        android:layout_below="@id/captureBtn"
        android:id="@+id/predictBtn"
        android:layout_marginTop="20dp"
        android:layout_centerHorizontal="true"/>

    <TextView
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:text="Result : "
        android:id="@+id/result"
        android:textSize="30dp"
        android:textAlignment="center"
        android:layout_below="@id/predictBtn"

        android:layout_marginTop="20dp" />


</RelativeLayout>
```

MainActivity Implementation

The MainActivity.java file in an Android app is typically responsible for managing the app's main activity, which is the primary screen that the user sees when they launch the app. It is typically responsible for setting up the app's layout and UI, handling user input and interactions, and performing other tasks such as loading data or making network requests. Here is an example of a simple MainActivity.java file for an image detection app:

Listing 2.2: activity$_{m}ain.xmllabel$

```
package com.example.imageclassify;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
```

```
 7
 8import android.Manifest;
 9import android.content.Intent;
10import android.content.pm.PackageManager;
11import android.graphics.Bitmap;
12import android.net.Uri;
13import android.os.Build;
14import android.os.Bundle;
15import android.provider.MediaStore;
16import android.view.View;
17import android.widget.Button;
18import android.widget.ImageView;
19import android.widget.TextView;
20
21import com.example.imageclassify.ml.MobilenetV110224Quant;
22
23import org.tensorflow.lite.DataType;
24import org.tensorflow.lite.support.image.TensorImage;
25import org.tensorflow.lite.support.tensorbuffer.TensorBuffer;
26
27import java.io.BufferedReader;
28import java.io.FileNotFoundException;
29import java.io.IOException;
30import java.io.InputStreamReader;
31
32
33public class MainActivity extends AppCompatActivity {
34   Button selectBtn, predictBtn, captureBtn;
35   TextView result;
36   ImageView imageView;
37   Bitmap bitmap;
38
39    @Override
40    protected void onCreate(Bundle savedInstanceState) {
41        super.onCreate(savedInstanceState);
42        setContentView(R.layout.activity_main);
43
44        getPermission();
45        String[] labels=new String[1001];
46        int cnt=0;
47        try {
48
49
50            BufferedReader bufferedReader = new BufferedReader(
                  new InputStreamReader(getAssets().open("labels.txt
                  ")));
```

```
51            String line = bufferedReader.readLine();
52            while (line!=null){
53                labels[cnt]=line;
54                cnt++;
55                line=bufferedReader.readLine();
56            }
57        }catch (IOException e){
58            e.printStackTrace();
59        }
60
61        selectBtn = findViewById(R.id.selectBtn);
62        predictBtn = findViewById(R.id.predictBtn);
63        captureBtn = findViewById(R.id.captureBtn);
64        result = findViewById(R.id.result);
65        imageView = findViewById(R.id.imageView);
66
67
68        selectBtn.setOnClickListener(new View.OnClickListener() {
69            @Override
70            public void onClick(View view) {
71                Intent intent = new Intent();
72                intent.setAction(Intent.ACTION_GET_CONTENT);
73                intent.setType("image/*");
74                startActivityForResult(intent, 10);
75
76
77            }
78        });
79
80  captureBtn.setOnClickListener(new View.OnClickListener() {
81      @Override
82      public void onClick(View view) {
83          Intent intent = new Intent(MediaStore.
                ACTION_IMAGE_CAPTURE);
84        startActivityForResult(intent,12);
85      }
86
87  });
88predictBtn.setOnClickListener(new View.OnClickListener() {
89    @Override
90    public void onClick(View view) {
91
92        try {
93            MobilenetV110224Quant model = MobilenetV110224Quant.
                newInstance(MainActivity.this);
94
```

```
95            // Creates inputs for reference.
96            TensorBuffer inputFeature0 = TensorBuffer.
                  createFixedSize(new int[]{1, 224, 224, 3},
                  DataType.UINT8);
97            bitmap = Bitmap.createScaledBitmap(bitmap, 224, 224,
                  true);
98            inputFeature0.loadBuffer(TensorImage.fromBitmap(
                  bitmap).getBuffer());
99
100           // Runs model inference and gets result.
101           MobilenetV110224Quant.Outputs outputs = model.process
                  (inputFeature0);
102
103           TensorBuffer outputFeature0 = outputs.
                  getOutputFeature0AsTensorBuffer();
104
105           result.setText(labels[getMax(outputFeature0.
                  getFloatArray())]+"");
106
107           // Releases model resources if no longer used.
108           model.close();
109       } catch (IOException e) {
110           // TODO Handle the exception
111       }
112
113
114   }
115});
116   }
117
118int getMax(float[] arr){
119   int max=0;
120       for(int i=0;i<arr.length;i++){
121
122           if(arr[i]>arr[max]){
123               max=i;
124           }
125       }
126       return max;
127}
128   void getPermission(){
129       if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
130           if(checkSelfPermission(Manifest.permission.CAMERA)!=
                  PackageManager.PERMISSION_GRANTED);
131           ActivityCompat.requestPermissions(MainActivity.this,
                  new String[] {Manifest.permission.CAMERA},11);
```

```
132
133         }
134
135     }
136
137     @Override
138     public void onRequestPermissionsResult(int requestCode,
            @NonNull String[] permissions, @NonNull int[]
            grantResults) {
139         if(requestCode==11){
140             if(grantResults.length>0){
141                 if(grantResults[0]!=PackageManager.
                        PERMISSION_GRANTED){
142                     this.getPermission();
143                 }
144
145             }
146         }
147         super.onRequestPermissionsResult(requestCode, permissions
                , grantResults);
148     }
149
150     @Override
151     protected void onActivityResult(int requestCode, int
            resultCode, @Nullable Intent data) {
152         if(requestCode==10){
153             if(data!=null){
154                 Uri uri = data.getData();
155                 try {
156                     bitmap = MediaStore.Images.Media.getBitmap(
                            this.getContentResolver(),uri);
157                     imageView.setImageBitmap(bitmap);
158                 } catch (FileNotFoundException e) {
159                     e.printStackTrace();
160                 } catch (IOException e) {
161                     e.printStackTrace();
162                 }
163
164             }
165         }
166         else if(requestCode==12){
167           bitmap = (Bitmap) data.getExtras().get("data");
168           imageView.setImageBitmap(bitmap);
169
170         }
171         super.onActivityResult(requestCode, resultCode, data);
```

```
172     }
173}
```

Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure

A simulation environment is a software tool or platform that is used to create and test virtual models or scenarios. In the context of an image detection app, a simulation environment might be used to test and validate the app's machine learning model and user interface before deploying it to real devices. Some benefits of using a simulation environment for image detection app development might include:

- Ability to test the app under a wide range of conditions: A simulation environment allows developers to test the app under a variety of different conditions and scenarios, including different types of images, lighting conditions, and device configurations. This can help to identify and fix problems with the app before it is released.
- Ability to test the app without real devices: A simulation environment allows developers to test the app without the need for physical devices, which can be expensive and time-consuming to obtain and maintain. This can save resources and reduce costs.
- Ability to simulate user interactions: A simulation environment allows developers to simulate user interactions with the app, such as taking pictures, selecting images, and making predictions. This can help to ensure that the app is easy to use and intuitive.

3.2 Results Analysis/Testing

3.2.1 UI/UX

After writing complete xml code it will look like this.

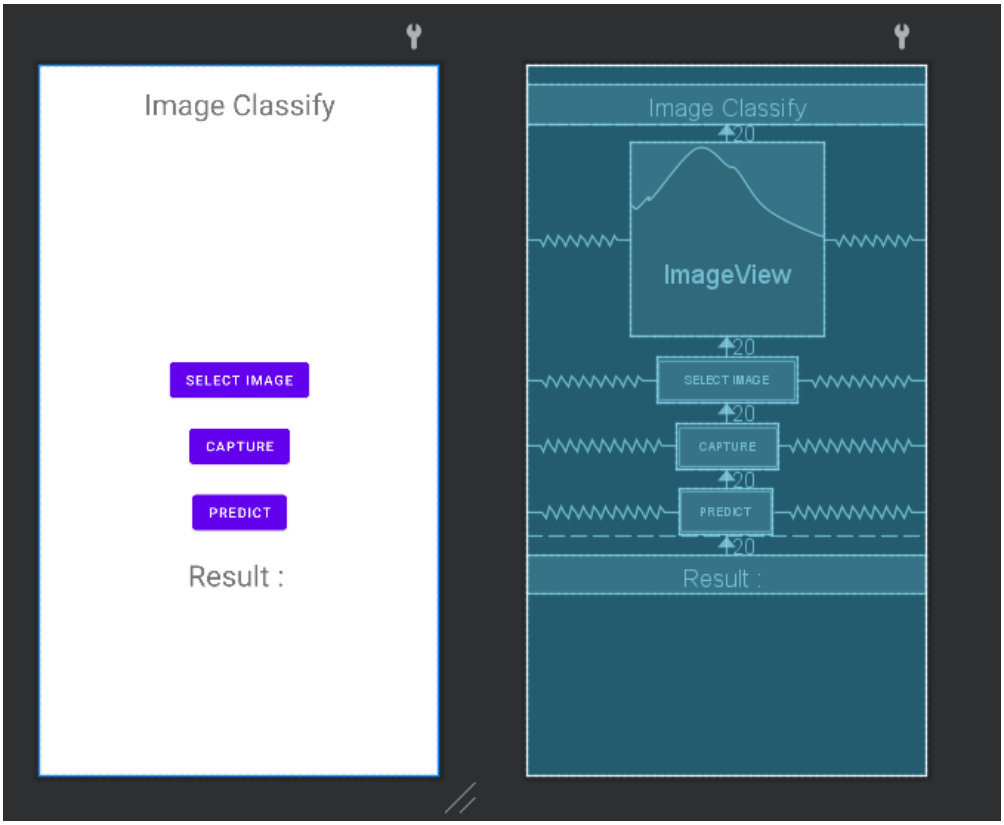

Figure 3.1: UI design

3.2.2 Final App Test

First open app

Select button will take it to the memory.Then select it and click the predict button.

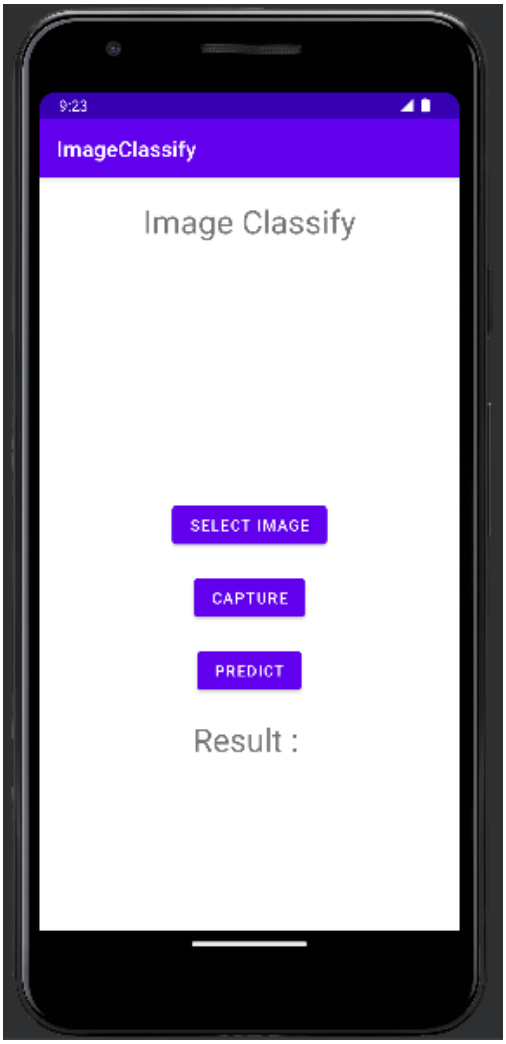

Figure 3.2: App Overview

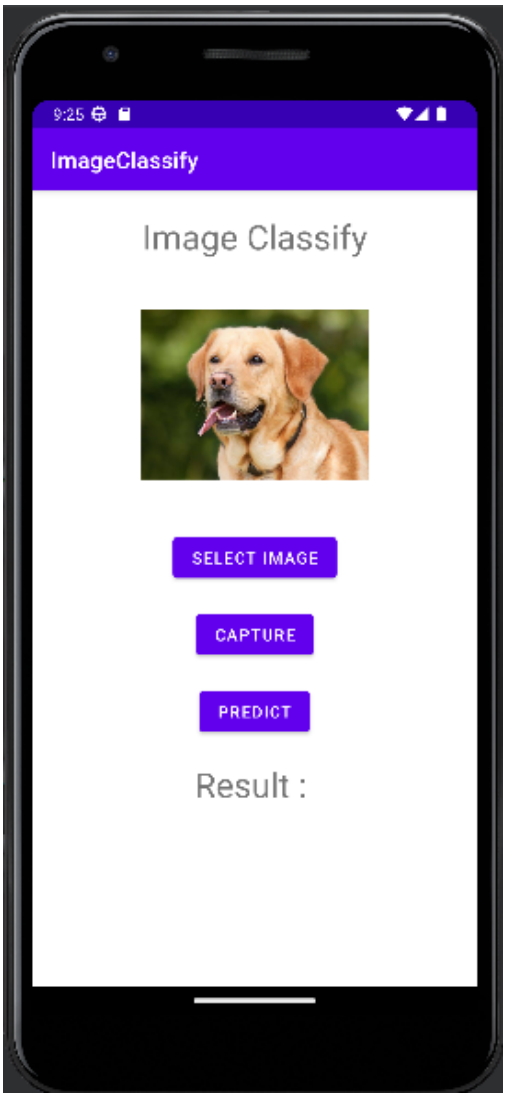

Figure 3.3: App overview

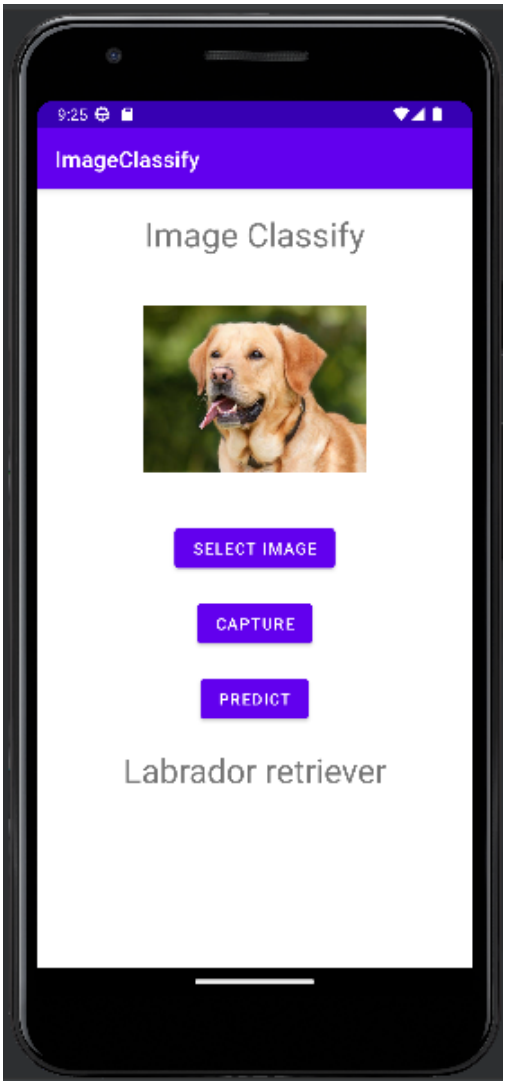

Figure 3.4: Detect picture

It will detect the picture

3.3 Results Overall Discussion

The results of an image detection app will depend on a number of factors, including the accuracy and performance of the machine learning model, the quality of the images that are used as input, and the specific goals and objectives of the app. To evaluate the results of an image detection app, developers can use a variety of metrics such as accuracy, speed, user experience, scalability, and robustness. Accuracy measures the ability of the app to correctly identify and classify objects in images, while speed measures the time it takes the app to process an image and make a prediction. User experience measures the ease of use and overall satisfaction of the app for the user, while scalability measures the app's ability to handle a large volume of images and work with a wide range of objects and image types. Robustness measures the app's ability to handle variations in lighting

and other environmental conditions and operate reliably over time. By evaluating the results of an image detection app using these and other metrics, developers can identify areas where the app is performing well and areas where it may need improvement, and can use this information to guide further development and ensure that the app meets the needs of the users.

Chapter 4

Conclusion

4.1 Discussion

In general, image detection apps can be useful tools for a wide range of purposes, including object recognition, facial recognition, text recognition, and more. These apps use machine learning algorithms to analyze images and extract information about their contents, providing a fast and convenient way to obtain information about objects and features in images. There are many tools and libraries available for building image detection apps, including machine learning frameworks such as TensorFlow and PyTorch, image processing libraries such as OpenCV and Pillow, and mobile development platforms such as Android Studio and Xcode. A simulation environment can also be a useful tool for developing and testing image detection apps, allowing developers to test the app under a wide range of conditions and scenarios without the need for physical devices. By evaluating the results of an image detection app using metrics such as accuracy, speed, user experience, scalability, and robustness, developers can identify areas where the app is performing well and areas where it may need improvement, and can use this information to guide further development and ensure that the app meets the needs of the users.

4.2 Limitations

There are several limitations to consider when developing an image detection app. Some of the main limitations include:

1. Accuracy: Machine learning models are not perfect, and there is always a possibility of errors or incorrect predictions. This can be particularly problematic for image detection apps, as small variations in the images or the objects being detected can have a big impact on the accuracy of the predictions.

2. Data quality: The quality of the data that is used to train the machine learning model can also have a big impact on the accuracy and performance

of the app. If the data is of poor quality or is not representative of the types of images that the app will be used with, the accuracy of the predictions may be impaired.

3. Performance: Image detection apps can be resource-intensive, and may require significant processing power and memory to run efficiently. This can be a limitation on devices with limited hardware resources, such as older smartphones or tablets.

4. Privacy concerns: Image detection apps may raise privacy concerns, as they may be used to collect and analyze images of people or objects. Developers should be mindful of these concerns and take steps to protect the privacy of users when developing and deploying image detection apps.

Overall, these are some of the main limitations to consider when developing an image detection app. By understanding these limitations and taking steps to address them, developers can help to ensure that the app is accurate, reliable, and respectful of user privacy.

4.3 Scope of Future Work

There is a wide range of potential future work that could be done with image detection apps. Some possible directions for future development include:

1. Improving accuracy: One area of focus could be on improving the accuracy of the machine learning models used in image detection apps. This could involve developing new algorithms or techniques for analyzing images and extracting relevant features, or using more advanced machine-learning frameworks or libraries to build and train the models.

2. Expanding the scope of objects and features that the app can detect: Another possibility is to expand the range of objects and features that the app can detect. This could involve adding support for new types of objects, such as animals or plants, or developing the ability to detect more subtle or complex features, such as facial expressions or body language [3].

3. Improving performance: Another area of focus could be on improving the performance of image detection apps. This could involve optimizing the machine learning models to run more efficiently on mobile devices, or developing techniques for reducing the amount of data that needs to be processed.

4. Enhancing the user experience: Another direction for future work could be to focus on improving the overall user experience of image detection apps. This could involve adding new features or functionality to the app, or designing more intuitive and engaging UI elements [4].

Overall, there are many potential directions for future work with image detection apps, depending on the specific goals and needs of the developers and users.

References

[1] Lamiaa A Elrefaei, Alaa Alharthi, Huda Alamoudi, Shatha Almutairi, and Fatima Al-rammah. Real-time face detection and tracking on mobile phones for criminal detection. In *2017 2nd International Conference on Anti-Cyber Crimes (ICACC)*, pages 75–80. IEEE, 2017.

[2] Fadwa Al-Azzoa, Arwa Mohammed Taqia, and Mariofanna Milanovab. Human related-health actions detection using android camera based on tensorflow object detection api. *International Journal of Advanced Computer Science and Applications*, 9(10), 2018.

[3] Dwi Sunaryono, Joko Siswantoro, and Radityo Anggoro. An android based course attendance system using face recognition. *Journal of King Saud University-Computer and Information Sciences*, 33(3):304–312, 2021.

[4] Siti Nurulain Mohd Rum and Fariz Az Zuhri Nawawi. Fishdetec: A fish identification application using image recognition approach. *International Journal of Advanced Computer Science and Applications*, 12(3), 2021.