Green University of Bangladesh

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

Animal Image Detecting Application

Course title: Artificial Intelligence Lab
Course Code: CSE 316 **Section:** 202 D2

Students Details

Name	ID
A.K.M. Atiqur Rahman	202002035

Submission Date : 18-06-2023
Course Teacher's Name: Ms. Fatema Akter

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

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

Contents

Chapter 1

Introduction

Animal Image Detecting Application is the title of the project. An animal image detection app is a android application that is able to analyze images and provide information about their contents. This application leveraging machine learning algorithms to analyze images and provide information about their contents. These apps employ techniques such as object recognition, facial recognition, and text recognition to identify various subjects within images. The process of building such an app entails training a machine learning model capable of recognizing the desired objects or features. Integration of the trained model into the app enables it to make accurate predictions regarding the contents of images. Image detection apps can be developed for different platforms, including iOS and Android, using various programming languages and tools. By harnessing the power of machine learning, these apps open up a wide range of possibilities for image analysis and understanding.

1.1 Overview

The overview of the animal image detecting application is as follows: The primary objective of this app is to serve as an educational tool for children, enabling them to learn about different objects by detecting and identifying them within images. The target audience for this app is children, who can benefit from the interactive and educational aspects of the image detection functionality. This app will provide users with the ability to select or capture an image using their device's camera. Utilizing a machine learning model, the app will then analyze the image to detect and identify objects present. This detection process will be initiated by the user through a "predict" button. This project will be developed using Android Studio, leveraging the Java programming language for Android app development. Additionally, a machine learning framework or library will be employed for training the object detection model. The development timeline for the project is planned to align with the duration of the university semester, ensuring timely completion of the app. As a university student project, no specific budget has been allocated for the app's development. It will primarily rely on freely available resources and tools. The main challenges in the project include obtaining a high-quality machine learning model for accurate object detection and ensuring the re- liability

and accuracy of the app's object detection functionality. These challenges will be addressed through rigorous experimentation, testing, and iterative improvements. Overall, by creating an engaging and educational app, children will have the opportunity to explore and learn about various objects through the exciting medium of image detection.

1.2 Motivation

There are various motivations that can drive the development of an image detection app, depending on the goals and requirements of both developers and users. Some potential motivations for building such an app are:

- One motivation could be to develop a useful and captivating app for a particular purpose, such as education or entertainment. For instance, an image detection app designed for children can facilitate interactive learning by introducing them to diverse objects, animals, or subjects.
- Another motivation involves solving a particular problem or fulfilling a specific need. For instance, an image detection app could aid visually impaired individuals in object identification or navigating their surroundings, enhancing their independence and accessibility.
- Building an image detection app can serve as a means to demonstrate the capabilities of new technologies, particularly machine learning algorithms. Such apps provide tangible examples of how machine learning can be applied to real-world scenarios and showcase its potential to enhance various domains.
- Developing an image detection app can also be motivated by the potential to generate revenue through advertising or in-app purchases. By incorporating advertisements or offering premium features within the app, developers can explore monetization strategies to support their efforts.

Ultimately, the motivation for constructing an image detection app depends on the specific objectives of developers and the target audience they aim to serve. Whether the focus is on education, problem-solving, technology demonstration, or monetization, understanding the underlying motivation helps drive the development process and shape the app's features and functionalities.

1.2.1 Problem Statement

The problem statement for an animal image detecting application is highly dependent on the unique goals and requirements of both the developers and the users. Here are some potential problem statements for an image detection app:

1. "How can we develop an engaging and educational tool that utilizes image detection to enhance children's understanding of various objects, fostering interactive learning experiences?"

2. "What approach can we take to leverage machine learning algorithms and image detection techniques in order to empower visually impaired individuals by providing reliable object identification and aiding their navigation in real-world environments?"

3. "How can we enhance the accuracy and efficiency of optical character recognition (OCR) within an image detection app, enabling users to extract text with high precision and reliability?"

4. "What strategies and algorithms can be implemented to create a real-time image detection app that demonstrates robustness and accuracy in identifying and classifying a wide range of objects in diverse environmental conditions?"

By refining the problem statements, developers can gain a clearer understanding of the specific challenges they need to tackle in order to create an effective and impactful image detection app. This focused approach allows them to prioritize the development of relevant features and functionalities that address the needs and expectations of the app's users.

1.2.2 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:

Attribute	Explanation	Addressing
Maintainability: Depth of knowledge required	The system requires specialized knowledge in multiple domains such as image processing, machine learning, computer vision, etc.	Establish a team with expertise in the relevant domains. Provide training and knowledge sharing opportunities.
Speed: Range of conflicting requirements	The system needs to process images in real-time, keeping up with the pace of the production line.	Use efficient machine learning models and parallel processing methods for optimization.
Scalability: Depth of analysis required	The system should handle a large volume of images and various types of industrial equipment.	Implement scalable infrastructure and train the machine learning model on diverse datasets.
Robustness: Depth of analysis required	The system should handle variations in lighting and environmental conditions.	Employ robust image preprocessing techniques and machine learning algorithms. Perform regular monitoring and maintenance.

Table 1.1: Addressing P attributes

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.

1.3 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.

1.4 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 involve several essential steps. The process begins with defining the problem the app aims to solve and establishing specific goals and objectives. This sets the foundation for guiding the design and development process, ensuring the app fulfills user needs. The next step is designing the user interface (UI), creating screens, layouts, and incorporating elements such as buttons, text fields, and images. The UI should prioritize ease of use and intuitiveness to enhance user interaction. Subsequently, a machine learning model is trained to detect and classify the desired objects or features. This typically involves utilizing a machine learning framework/library like TensorFlow to build and train the model. Once trained, the model can be integrated into the app using the appropriate machine learning framework or library, such as using the TensorFlow Lite Interpreter for on-device inference or a server-based solution for making predictions. Finally, rigorous testing and debugging are conducted to ensure the app functions correctly and aligns with user requirements.

2.2 Project Details

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: $\text{activity}_{m}ain.xmllabel$

```
1<?xml version="1.0" encoding="utf-8"?>
2<RelativeLayout xmlns:android="http://schemas.android.com/apk/
    res/android"
3    xmlns:app="http://schemas.android.com/apk/res-auto"
```

```
4    xmlns:tools="http://schemas.android.com/tools"
5    android:layout_width="match_parent"
6    android:layout_height="match_parent"
7    android:background="@drawable/p"
8    android:scaleType="centerCrop"
9    tools:context=".MainActivity">
10
11   <TextView
12       android:id="@+id/title"
13       android:layout_width="match_parent"
14       android:layout_height="wrap_content"
15       android:layout_marginTop="20dp"
16       android:text="Image Detection"
17       android:textColor="#FFC107"
18       android:textAlignment="center"
19       android:textSize="30dp" />
20
21   <ImageView
22       android:layout_width="180dp"
23       android:layout_height="180dp"
24       android:id="@+id/imageView"
25       android:layout_below="@id/title"
26       android:layout_centerHorizontal="true"
27       android:layout_marginTop="20dp"
28       android:background="@drawable/r"/>
29
30
31   <Button
32       android:layout_width="wrap_content"
33       android:layout_height="wrap_content"
34       android:text="Select Image"
35       android:layout_below="@id/imageView"
36       android:id="@+id/selectBtn"
37       android:layout_marginTop="20dp"
38       android:layout_centerHorizontal="true" />
39
40   <Button
41       android:layout_width="wrap_content"
42       android:layout_height="wrap_content"
43       android:text="capture"
44       android:layout_below="@id/selectBtn"
45       android:id="@+id/captureBtn"
46       android:layout_marginTop="20dp"
47       android:layout_centerHorizontal="true" />
48
49   <Button
```

```
50          android:layout_width="wrap_content"
51          android:layout_height="wrap_content"
52          android:text="predict"
53          android:layout_below="@id/captureBtn"
54          android:id="@+id/predictBtn"
55          android:layout_marginTop="20dp"
56          android:layout_centerHorizontal="true" />
57
58      <TextView
59          android:layout_width="match_parent"
60          android:layout_height="40dp"
61          android:text="Result : "
62          android:textColor="#DD0808"
63          android:id="@+id/result"
64          android:textSize="30dp"
65          android:textAlignment="center"
66          android:layout_below="@id/predictBtn"
67          android:layout_marginTop="20dp" />
68
69      <TextView
70          android:layout_width="match_parent"
71          android:layout_height="wrap_content"
72          android:text="About"
73          android:id="@+id/aboutTitle"
74          android:textSize="24sp"
75          android:textColor="#C1D6AF"
76          android:textAlignment="center"
77          android:layout_below="@id/result"
78          android:layout_marginTop="40dp" />
79
80      <TextView
81          android:id="@+id/aboutDescription"
82          android:layout_width="match_parent"
83          android:layout_height="wrap_content"
84          android:layout_below="@id/aboutTitle"
85          android:layout_marginTop="-3dp"
86          android:text="Developer\nName: Prangon and Manik \n@copy
               right 2023"
87          android:textAlignment="center"
88          android:textSize="20sp"
89          android:textColor="#CDDC39" />
90
91  </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$_{m}ain.xmllabel$

```
1 package com.example.imageclassify;
2
3 import androidx.annotation.NonNull;
4 import androidx.annotation.Nullable;
5 import androidx.appcompat.app.AppCompatActivity;
6 import androidx.core.app.ActivityCompat;
7
8 import android.Manifest;
9 import android.content.Intent;
10 import android.content.pm.PackageManager;
11 import android.graphics.Bitmap;
12 import android.net.Uri;
13 import android.os.Build;
14 import android.os.Bundle;
15 import android.provider.MediaStore;
16 import android.view.View;
17 import android.widget.Button;
18 import android.widget.ImageView;
19 import android.widget.TextView;
20
21 import com.example.imageclassify.ml.MobilenetV110224Quant;
22
23 import org.tensorflow.lite.DataType;
24 import org.tensorflow.lite.support.image.TensorImage;
25 import org.tensorflow.lite.support.tensorbuffer.TensorBuffer;
26
27 import java.io.BufferedReader;
28 import java.io.FileNotFoundException;
29 import java.io.IOException;
30 import java.io.InputStreamReader;
31
32
33 public 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.

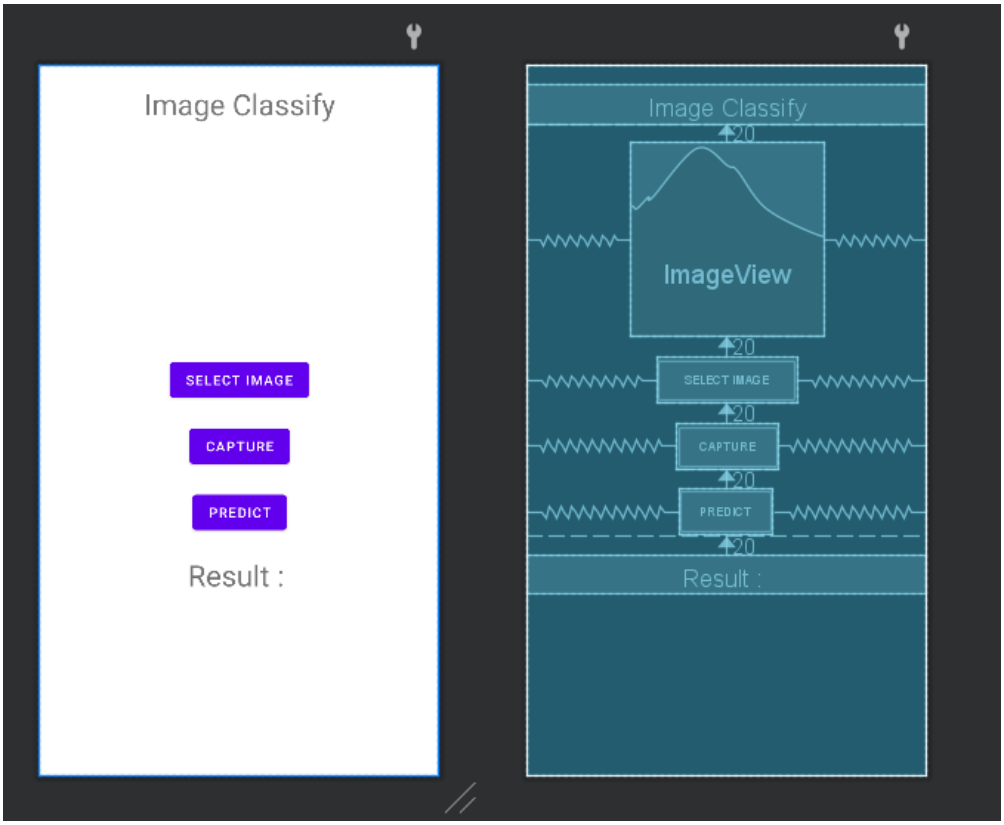

Figure 3.1: UI design

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.

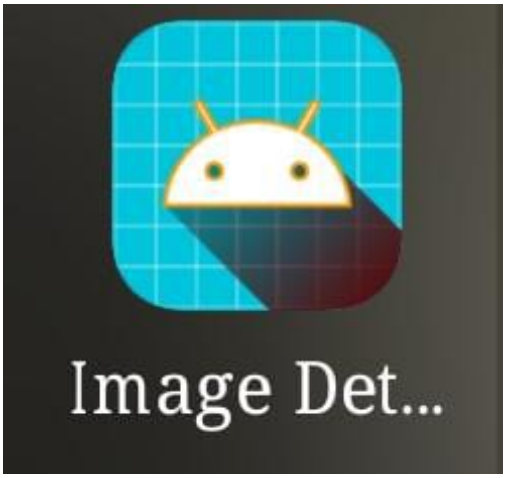

Figure 3.2: Image Detection APK

Figure 3.3: App Overview

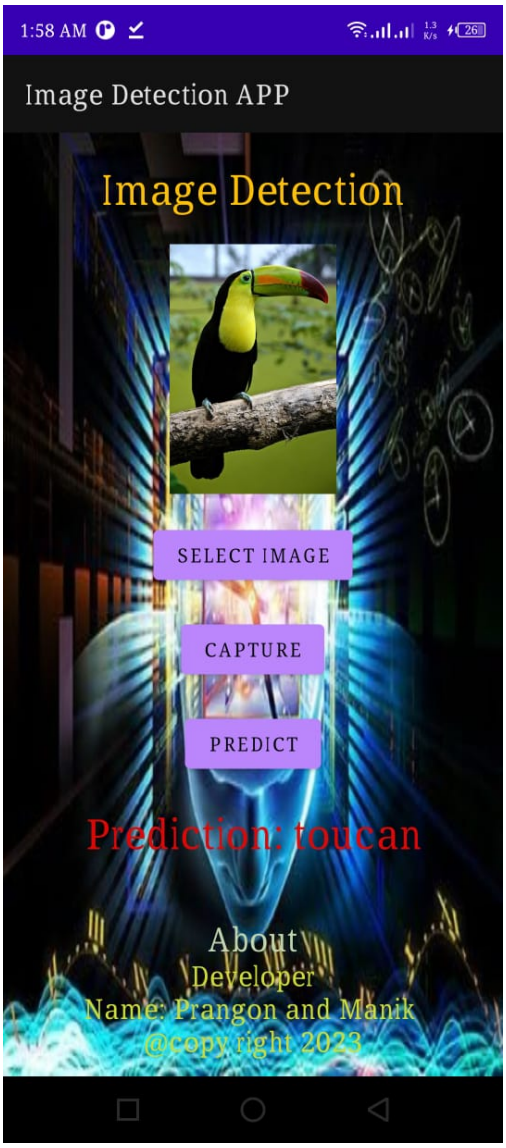

Figure 3.4: Detect a bird image

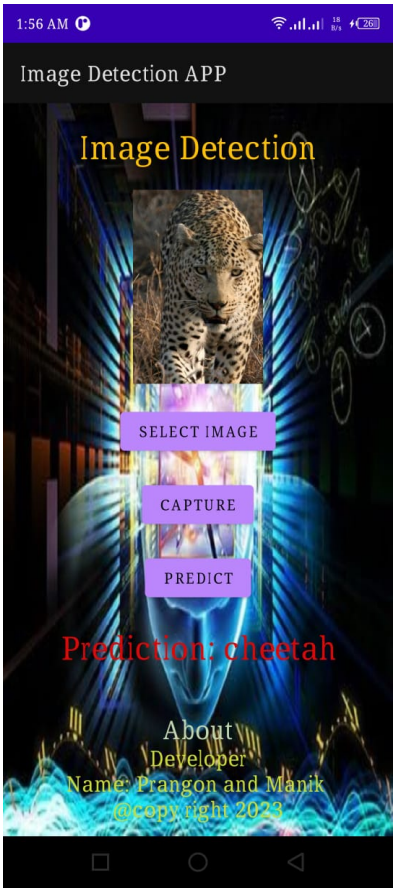

Figure 3.5: Detect a cheetah image

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, Animal Image detection android apps serve a variety of purposes, ranging from object recognition to facial recognition and text recognition. By utilizing machine learning algorithms, these apps analyze images and extract valuable information about their contents, offering a quick and convenient means of acquiring insights about various objects and features present in the images. To build such apps, developers can leverage a wide array of tools and libraries, including TensorFlow and PyTorch for machine learning, OpenCV and Pillow for image processing, and Android Studio and Xcode for mobile development. Additionally, employing a simulation environment during the app's development and testing phases can prove invaluable. This environment allows developers to thoroughly evaluate the app under diverse conditions and scenarios without the need for physical devices. By assessing key metrics such as accuracy, speed, user experience, scalability, and robustness, developers gain valuable insights into the app's performance. These insights can then guide further development efforts, ensuring that the app effectively caters to the needs of its 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 [1].

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 [2].

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] Dwi Sunaryono, Joko Siswantoro, and Radityo Anggoro. An android based course attendance system using face recognition. volume 33, pages 304–312. Elsevier, 2021.

[2] 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.