सहायक

(Sahaayak - Assistant)

Real-Time Object Detection App
Using Machine Learning

ECE 558 – Final Project

6th December 2018

Mohak Patel

Archit Tatwawadi

Maseeh College of Engineering
and Computer Science
PORTLAND STATE UNIVERSITY

Table of Content:

Project Overview

Objective:

To implement the android app to detect the real-time object using machine learning for the non-native user to learn the English language.

Motivation:

This project revolves around the problems faced by non-native English speakers. Many times, non-native people get stuck while describing a particular object. That is where our app comes to the rescue by showing the label name for the particular object in real-time.

Description:

- We implemented an android app which opens a camera and when pointed at a particular object, it displays its English name.
- The app starts with a splash screen. This app will then be secured with a login screen. When the user successfully logs in, it will open a camera which the user can use to point at an unknown object. A sign-up page for the new user to register.
- The app connects with the firebase ML (Machine Learning) kit to fetch the trained ML model. It can detect 1000 labels of a different object (Ex. Food, Shoe, Jeans etc.).
- The ML kit returns the object name which displayed on the screen.
- This trained ML model will be hosted using the Firebase ML kit.
- Android capabilities which are used for this app: Camera, Webservices (Firebase), ML kit.
- Tools required to implement this app is Android SDK.

Block Diagram

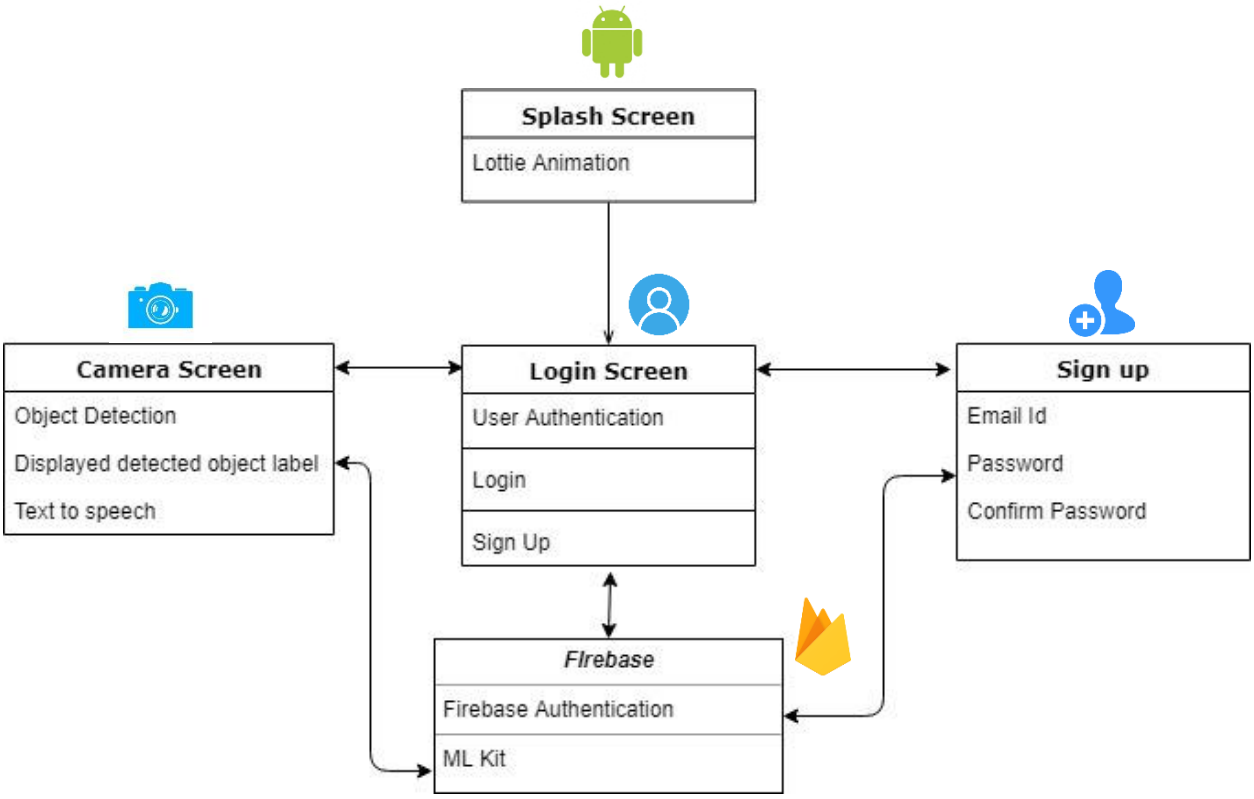

Figure 1: App block diagram

Design and Implementation

The application is divided into four main activity: Splash, Login, Sign Up, and Camera.

Splash screen:

A splash screen contains the Lottie animation which is loading animation in the .json format for the app. An animation is played for 3 seconds and after that login, Activity will open using the intent method.

The snippet of code to hide the system notification bar for full-screen utilization.

```
getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCREEN);
```

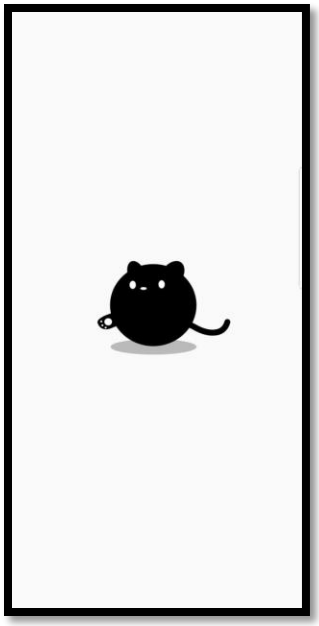

Figure 2: Animated Splash Screen

Lottie animation requires androidx module. To implement that we migrated from android to androidx. An androidx is a better version of android with more new features. Main advantage of androidx is that it supports the backword compatibility. We added the required dependencies of androidx and Lottie in the build. gradle (App module).

```
implementation 'androidx.appcompat:appcompat:1.0.0'
implementation 'androidx.constraintlayout:constraintlayout:2.0.0-alpha2'
implementation 'androidx.annotation:annotation:1.0.0'
androidTestImplementation 'androidx.test:runner:1.1.0-alpha3'
androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.0-alpha4'
implementation "com.airbnb.android:lottie:2.8.0"
```

Login screen:

Login screen contains different elements like buttons, editText field, TextView, gradient background, progress bar, and a vector image. We created different layout files for this element.

The snippet of codes to implement the gradient background and buttons background:

```
<gradient
    android:angle="0"
    android:endColor="@color/gradStop"
    android:startColor="@color/gradStart"
    android:type="linear" />
```

```
<corners
    android:bottomLeftRadius="20dp"
    android:bottomRightRadius="20dp"
    android:topLeftRadius="20dp"
    android:topRightRadius="20dp" />

<stroke
    android:width="1dp"
    android:color="@color/white" />
```

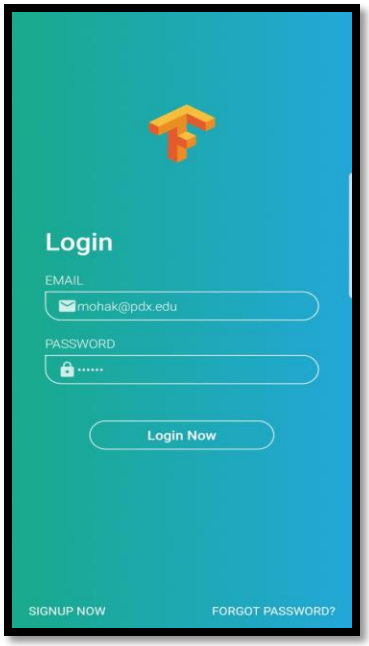

Figure 3: Login Screen

For security purpose, we use used the firebase authentication to verify the user's identity. To authenticate the user with firebase authentication, created the instance of firebase authentication and verify the user by passing the email and password to the signInWithEmailAndPassword:

```
mAuth.signInWithEmailAndPassword(email, password).addOnCompleteListener((task) → {
        progressBar.setVisibility(View.GONE);
        if (task.isSuccessful()) {
            finish();
            Intent intent = new Intent( packageContext: loginActivity.this, CameraActivity.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(intent);
        } else {
            makeText(getApplicationContext(), task.getException().getMessage(), Toast.LENGTH_SHORT).show();
        }
});
```

If email and password are correct, it opens the camera activity for the object detection else it will show the error for invalid email or password. To open the camera activity, we used an intent method.

Sign Up screen (Stretch Goal):

A sign-up screen contains three main field email id, password and confirms the password. If email id format does not match then it will show the error. Password must be more than 6 latter and it should match with the confirmation password to register on the app using firebase authentication.

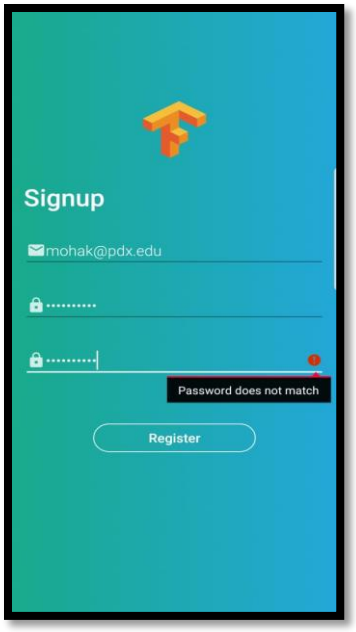

Figure 4: Sign Up Screen

The snippet of code to sign up for the new user:

```
mAuth.createUserWithEmailAndPassword(email, password).addOnCompleteListener( activity: this,(task) → {
        progressBar.setVisibility(View.GONE);
        if (task.isSuccessful()) {
            //  finish();
            startActivity(new Intent( packageContext: signupActivity.this, loginActivity.class));
```

If the user already registered and again try to register then it will toast the message that user is already registered. To handle this exception, we used the FirebasebaseAuthUserCollisionException method. For errors, we used the seterror class.

Camera screen:

For the real-time object detection, we setup the camera preview on screen, firebase ML kit for object detection, Overlay method to show the label on screen and text to speech synthesis for the label. For that purpose, we added few permission in AndroidManifest.xml file.

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.INTERNET"/>
<uses-feature android:name="android.hardware.camera.autofocus" />
```

Camera Preview Setup:

The camera opens in a background thread, so that main thread will not overload (Ex. Handling of large pixel arrays will hinder other events).

Figure 5: Camera Screen

To run the camera on a background thread, we used the HandlerThread. According to the android documentation, HandlerThread is a handy class for starting a new thread that has a looper. The looper can then be used to create handler classes and start() must still be called. The snippet of code:

```
private void openBackgroundThread() {
    backgroundThread = new HandlerThread( name: "camera_background_thread");
    backgroundThread.start();
    backgroundHandler = new Handler(backgroundThread.getLooper());
}
```

And we invoke the background thread in OnResume() method.

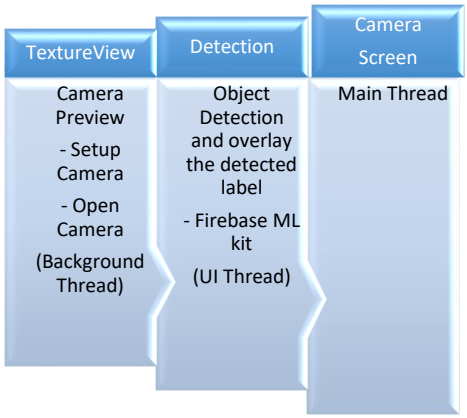

Figure 6: Threading

To access the camera, called the camera2 API. It requires the texture view to access the camera. By defining the camera facing (front or back) and surfaceTextureListener, we open the camera for object detection. The snippet of code to access the camera2 API:

```
cameraManager = (CameraManager) getSystemService(CAMERA_SERVICE);
//cameraFacing = CameraCharacteristics.LENS_FACING_BACK;

surfaceTextureListener = new TextureView.SurfaceTextureListener() {
    @Override
    public void onSurfaceTextureAvailable(SurfaceTexture surfaceTexture, int width, int height) {
        setUpCamera();
        openCamera();
    }
```

Firebase ML Kit:

ML kit available for image labeling: 1. On Device 2.Cloud base. The main advantage of ML kit is, it doesn't require additional training. We are using the on-device firebase ML kit for object detection. So when user will download the app, it will automatically download the ML model from the firebase for image labeling.

Pros and Cons of on Device ML model:

- Faster than cloud base
- Less number of object labels
- Required more memory on the device (~ 33 Mb)

To setup the ML kit, need to follow the few steps:

Step 1: For the object detection using the firebase ML kit, first add the ML dependencies in build.gradle.

```
implementation 'com.google.firebase:firebase-ml-vision:18.0.1'
implementation 'com.google.firebase:firebase-ml-vision-image-label-model:17.0.2'
```

Step 2: Declaration required to configure the app to automatically download the ML model to the device after your app is installed from the Play Store.

```
<meta-data
    android:name="com.google.firebase.ml.vision.DEPENDENCIES"
    android:value="label" />
```

Step 3: we create the FirebaseLabelDetector object for the label detection of the object. Which set the threshold for object detection confidence. In this case, it is 0.8.

```
FirebaseVisionLabelDetectorOptions options =
        new FirebaseVisionLabelDetectorOptions.Builder()
                .setConfidenceThreshold(0.8f)
                .build();
```

Step 4: After that, we create the FirebaseVisionImage object to read the bitmap image captured by the camera and created the FirebaseVisionLabelDetector instance.

```
FirebaseVisionImage image = FirebaseVisionImage.fromBitmap(bitmap);

FirebaseVisionLabelDetector detector = FirebaseVision.getInstance()
        .getVisionLabelDetector(options);
```

Step 5: We pass that image to the detectInImage method to get the label for the object. Which return the label for the detected object.

```
Task<List<FirebaseVisionLabel>> result =
        detector.detectInImage(image)
                .addOnSuccessListener(
                        new OnSuccessListener<List<FirebaseVisionLabel>>() {
                            @Override
                            public void onSuccess(List<FirebaseVisionLabel> labels) {
```

After getting the top 10 labels of the objects from the image, we overly the first label on camera preview, which has the highest confidence.

Text To speech (Stretch Goal):

To speak out the detected object label name, we implemented the onTouch () method. So that when user will touch on the screen, it will speak out the label name using TextToSpeech class.

Android allows us to convert text to speech. First, instantiate the object of TextToSpeech class and then specify the initListener. In that listener, first set the language.

```
t1=new TextToSpeech(getApplicationContext(), (status) → {
        if(status != TextToSpeech.ERROR) {
            t1.setLanguage(Locale.ENGLISH);
        }
});
```

After that using the method **speak,** converts the text to speech. We can also set the speaking speed and pitch.

```
private void speak() {
    String toSpeak = toplables.get(0);
    t1.setPitch(1.3f);
    t1.setSpeechRate(1f);
    t1.speak(toSpeak, TextToSpeech.QUEUE_FLUSH,  params: null, utteranceId: null);
}
```

Challenges

- Android to Androidx migration: When we build the gridle file, it automatically generates the android dependencies every time. So it conflicts with the android dependencies. It took time to figure out that conflicting dependencies. So every time when we build the gridle file or create a new activity, we need to delete the conflicting dependencies.
- Implementation of camera2 API: Camera2 API uses the textureview and preview the image on texureview. We need to set the height and weight of the camera preview properly otherwise it gets too compressed or stretched. And it affects the object detection part and doesn't give the proper results.
- Setup the real-time object detection: To make real-time object detection, we need to capture the images continuously for object detection. After capturing the image, it took some time to get the label. So for that amount of time, we dropped the captured frames. When we captured the image, first we checked the textureview. If it is not available then we dropped that frame and capture the image only when texureview is available.

Results

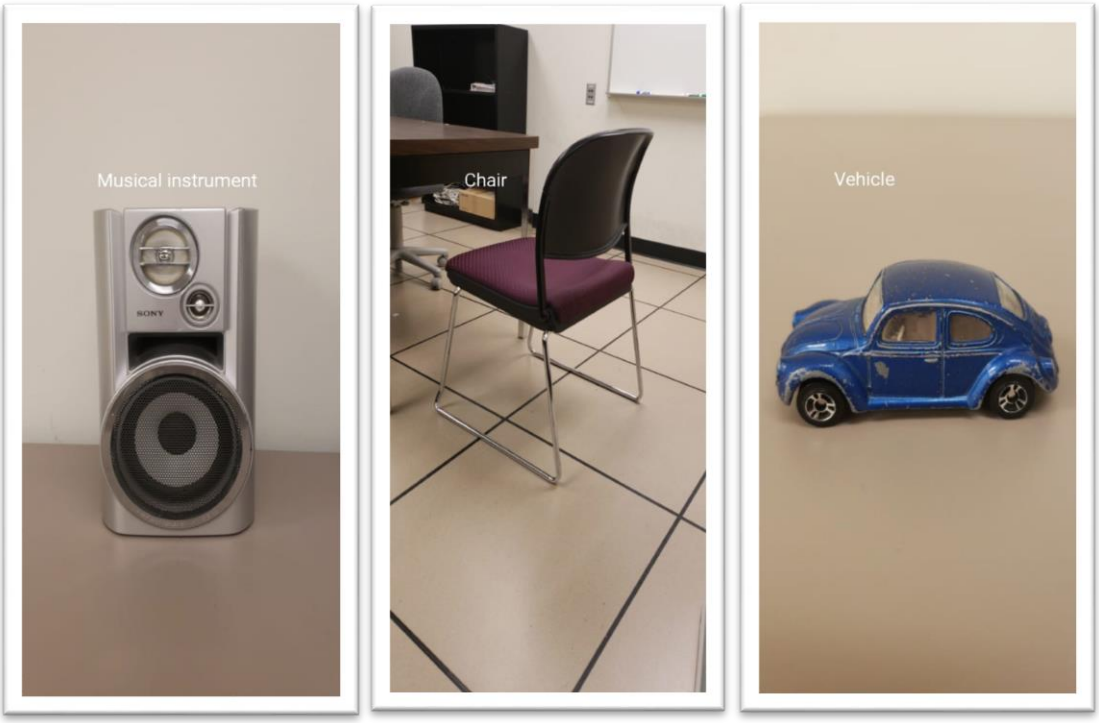

Figure 7: Detected Objects

Features

- Animated splash screen
- Secured login
- User registration page
- Real-time object detection
- Detect 1000 different labels
- The label on camera preview in the English language
- Speak out the label name in the English language

Future Scope

- For login using a smart lock like Face recognition.
- label in native and English languages
- Support more language
- The box around the detected object
- Add more labels for detection

User Reviews

User 1: App works perfectly but needs support for more languages.

User 2: When the label is displayed, it should display the label in native and English languages.

References

[1] Lecture notes – Prof. Roy Kravitz
[2] https://firebase.google.com/docs/auth/
[3] https://firebase.google.com/docs/ml-kit/android/label-images#on-device
[4] http://stephendnicholas.com/posts/android-handlerthread
[5] https://developer.android.com/reference/android/speech/tts/TextToSpeech
[6] Big Nerd Ranch Guide, 3rd Edition – Bill Phillips, Chris Stewart, and Kristen Marsicano