# <u>REALTIME OBJECT DETECTION AND CLASSIFICATION</u>

**A project report submitted in partial fulfilment of the requirements for the award of**

**the degree of**

**Bachelor of Technology**

**in**

**Computer Science and Engineering**

**By**

ARBAZ AHMED KHAN (Roll No. 25500118057)
ASWINI KUMAR SAHOO (Roll No. 25500118054)
CHANDRIMA ROY (Roll No. 25500118050)
KISHOR YADAV (Roll No. 25500118039)
RISHABH RAJ GUPTA (Roll No. 25500118027)

**Under the Guidance of**

**Mr. Anjan Kumar Payra**

**Assistant Professor, Department of CSE**

**at**

**DR. SUDHIR CHANDRA SUR INSTITUTE OF TECHNOLOGY AND SPORTS COMPLEX**

(*Permanently Affiliated to MAKAUT, Approved by AICTE*)

540, Dum Dum Rd, near Dum Dum, Surer Math, Dum Dum, Kolkata, West Bengal 700074

# CERTIFICATE

This is to certify that this is a bona fide record of the final year project entitled “***RealTime Object Detection and Classification***” done satisfactorily at ***Dr. Sudhir Chandra Sur Institute of Technology and Sports Complex*** by ***Arbaz Ahmed Khan, Aswini Kumar Sahoo, Chandrima Roy, Kishor Yadav and Rishabh Raj Gupta*** in partial fulfilment of the requirements for the award of the degree of ***Bachelor of Technology in Computer Science and Engineering***.

**Date: ________________**

**Mr. Anjan Kumar Payra**
**Assistant Professor, Department of CSE**
**(Signature of the Project Guide)**

**Mrs. Rinku Supakar**
**(Signature of the HOD)**
**Department of Computer Science and Engineering**

**(Signature of the Examiner)**

## ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide **Mr. Anjan Kumar Payra**, Assistant Professor, Department of Computer Science and Engineering, DSCSITSC, for his guidance with unsurpassed knowledge and immense encouragement. We are grateful to **Mrs. Rinku Supakar**, Head of the Department, Computer Science and Engineering, for providing us with the required facilities for the completion of the project work.

We are very much thankful to the Principal and Management, DSCSITSC, **Dr. Saradindu Panda**, for their encouragement and cooperation to carry out this work.

We express our thanks to all teaching faculty of the Department of CSE, whose suggestions during reviews helped us in accomplishment of our project. We would like to thank all non-teaching staff of the Department of CSE, DSCSITSC for providing great assistance in the accomplishment of our project. We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. At last but not the least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

**Team Members :**

Arbaz Ahmed Khan
Aswini Kumar Sahoo
Chandrima Roy
Kishor Yadav
Rishabh Raj Gupta

# CONTENTS

# CHAPTER 1

# INTRODUCTION

## 1.1 Introduction

A few years ago, the creation of software and hardware image processing systems was mainly limited to the development of user interfaces, which most of the programmers of each firm were engaged in. The situation has changed significantly with the advent of the Windows operating system when most developers started to solve the problems of image processing themselves. However, this has not yet led to cardinal advances in solving typical facial recognition tasks,
car numbers, traffic signs, remote and medical image analysis, etc. Each of these "eternal" problems is solved through trial and error through the effort of numerous groups of engineers and scientists. Since modern technical solutions are too expensive, the task of automating the creation of software tools for solving intellectual problems is formulated intensively and solved externally. In the field of image processing, the required tools must support image analysis and recognition of previously unknown content and ensure effective application development by ordinary programmers. Just like the Windows Toolkit supports the creation of interfaces to solve various applied problems.

Object recognition is delineated as a group of related computer vision tasks that involve activities like identifying objects in digital photographs. Image classification involves activities like predicting the category of objects in an image. Object localization refers to identifying the location of one or more objects in an image and drawing a bounding-box around their extent. Object detection does the work of integrating these two tasks and localizes and classifies one or more objects in an image. When a user or professional refers to the term “object recognition”, they usually mean “object detection”. It's going to be challenging for beginners to completely differentiate between different computer vision tasks.

One of the further extensions of this breakdown of computer vision tasks is object segmentation, also called "object instance segmentation" or "semantic segmentation", where recognized objects' instances are displayed by highlighting specific pixels of the object instead of a thick bounding box. From this breakdown we can understand that object recognition refers to a suite of challenging computer vision tasks. Humans can sight and determine objects present in an image. The human sensory system is quick and accurate and may conjointly perform complex tasks like distinguishing multiple objects and detect obstacles with little conscious thought. With the provision of enormous sets of data, quicker GPUs, and better algorithms, we can now easily train computers to detect and classify multiple objects within an image with high accuracy. Image classification also involves assigning a class label to an image, while object localization involves drawing a bounding box around one or more objects in an image. Object detection is always more and more challenging and combines these two tasks and draws a bounding box around each object of interest in the image and assigns it a class label. Collectively, all of these problems are known as object recognition.

## 1.2 Motivation

Real-time object detection is a very demanding task on its own. Modern computers can run object detection models without a break, with latencies in nanoseconds. Although we have come across a long way technologically we still struggle with real-time object detection on mobile phones. But nowadays a large proportion of mobile phones have decent processing power, and with the latest advancements now we can detect objects in mobile phones with lower latencies in single digits, we can even detect multiple objects with decent accuracy. But it comes with a caveat, while we can detect a wide range of objects, it can only classify them coarsely.

Since the applications of real-time object detection are overwhelmingly increasing we decided to come up with a prototype of our own, demonstrating the fundamentals of object detection and a workaround for classification in mobile devices.

# CHAPTER 2

# RELATED WORKS

## 2.1 State of Modern Real-time Object Detection

Real-time object detection and tracking is a vast, vibrant yet inconclusive and complex area of computer vision. Due to its increased utilization in surveillance, tracking systems used in security and many other applications have propelled researchers to continuously devise more efficient and competitive algorithms. However, problems emerge in implementing object detection and tracking in real-time; such as tracking under dynamic environment, expensive computation to fit the real-time performance, or multi-camera multi-objects tracking make this task strenuously difficult.

In the last few years, the rapid advances of deep learning techniques have greatly accelerated the momentum of object detection. With deep learning networks and the computing power of GPU's, the performance of object detectors and trackers has greatly improved, achieving significant breakthroughs in object detection.

Machine learning (ML) is a branch of artificial intelligence (AI), and it essentially involves learning patterns from examples or sample data as the machine accesses the data and has the ability to learn from it (supervised learning on annotated images). Deep Learning is a specialized form of machine learning which involves learning in different stages.

A wide range of computer vision applications has become available for object detection and tracking. As a result, numerous real-world applications, such as healthcare monitoring, autonomous driving, video surveillance, anomaly detection, or robot vision, are based on deep learning object detection.

- **Object detection in Retail**

Strategically placed **people counting systems** throughout multiple retail stores are used to gather information about how customers spend their time and customer footfall. AI-based customer analysis to detect and track customers with cameras helps to gain an understanding of customer interaction and customer experience, optimize the store layout, and make operations more efficient. A popular use case is the detection of queues to reduce waiting time in retail stores.

- **Autonomous Driving**

Self-driving cars depend on real-time object detection to recognize pedestrians, traffic signs, other vehicles, and more. For example, Tesla's Autopilot AI heavily utilizes object detection to perceive environmental and surrounding threats such as oncoming vehicles or obstacles.

- **Animal detection in Agriculture**

Object detection is used in agriculture for tasks such as counting, animal monitoring, and evaluation of the quality of agricultural products. Damaged produce can be detected while it is in processing using machine learning algorithms.

- **People detection in Security**

A wide range of security applications in video surveillance are based on object detection, for example, to detect people in restricted or dangerous areas, suicide prevention, or to automate inspection tasks on remote locations with computer vision.
Example of object detection in **video analytics** for people detection in dangerous areas using CCTV cameras

- **Medical feature detection in Healthcare**

Object detection has allowed for many breakthroughs in the medical community. Because medical diagnostics rely heavily on the study of images, scans, and photographs, object detection involving CT and MRI scans has become extremely useful for diagnosing diseases, for example with ML algorithms for tumour detection.

Object detection methods can be categorized into two main types: One-stage vs. two-stage object detectors. In general, deep learning-based object detectors extract features from the input image or video frame. An object detector solves two subsequent tasks:

1. Detecting possible object regions.
2. Classifying the image in those regions into object classes.

To simplify the process, you can separate those tasks into two stages. Other methods combine both tasks into one step (single-stage detectors) to achieve higher performance at the cost of accuracy.

Two-stage detectors: In two-stage object detectors, the approximate object regions are proposed using deep features before these features are used for the classification as well as bounding box regression for the object candidate.

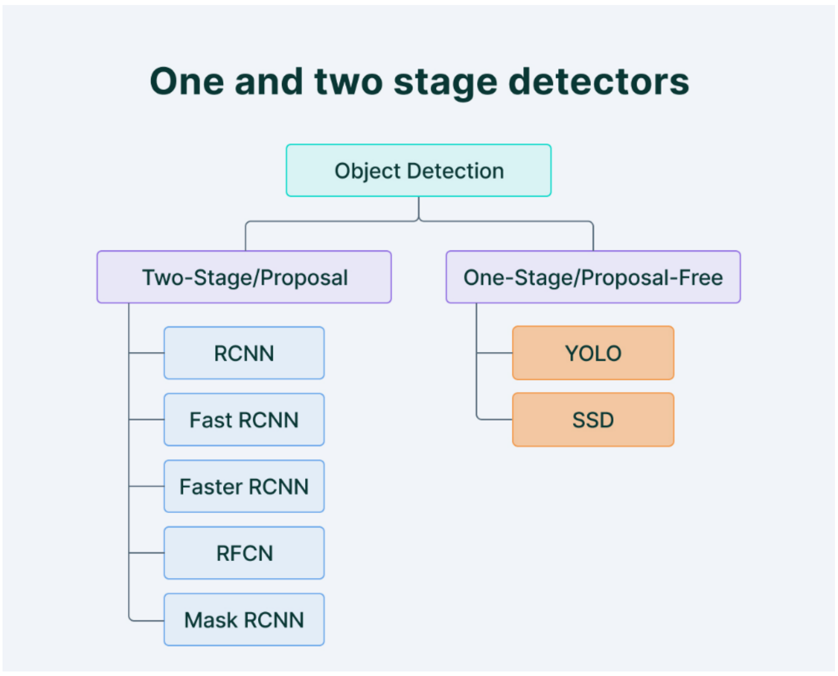


Figure 1

Despite all the advancements in Object Detection, only a few advancements have been made when it comes to mobile applications using real-time object detection. Some of them are mentioned below.

## 2.2 Object Detector by AnhNguyen

This app has 2 modes:

- **Detection mode:** This mode enables detection of 80 classes from COCO dataset with bounding box.
- **Classification mode:** This mode can perform identification of 1000 classes from ImageNet dataset without bounding box.

*To start and stop live preview in the application, just touch on the preview screen.

The Deep Learning software library TensorFlow and 2 MobileNet models are used for detection and classification. This app is just a demo of using Machine Learning with TensorFlow on Android devices. It uses the YOLO model for object detection. No internet connection is required for using this app.

**Drawbacks :**

- Latency is quite high (ranging from 250-1000ms) depending on the device it is running on.
- The size of application is quite large
- Since it is based on One-Stage detection, accuracy is lower.

## 2.3 Google Lens

Google Lens is an AI-powered technology that uses your smartphone camera and deep machine learning to not only detect an object in front of the camera lens, but understand it and offer actions such as scanning, translation, shopping, and more. It enables you to point your phone at something, such as a specific flower, and then ask Google Assistant what the object you're pointing at is. You will get the results.

As of now, Google lens can detect multiple objects from the image, but it does all the operations of detection and classification after the image is being clicked by the user on the smartphone. It does not provide the feature of real-time object detection.

Also, for getting results in our smartphone, Google lens requires a good internet connectivity in our device, surpassing which will give no output.

The processing of information and everything is being done on cloud, after the image is being clicked by the user. So, all our data is being taken into the cloud, thereby raising a concern for privacy.

# CHAPTER 3

# PRESENT WORK

## 3.1 Introduction

The aim of object detection in our project is to detect all instances of objects from a known class, such as people, cars or any sort of object in an image. Generally, only a small number of instances of the object are present in the image, but there is a very large number of possible locations and scales at which they can occur and that need to somehow be explored. Each detection of the image in our application is reported with basic information. This is as simple as the location of the object, a location and scale, or the extent of the object defined in terms of a bounding box. An example of object detection in an image from our application that specifies the objects shown in the image in Figure 2.

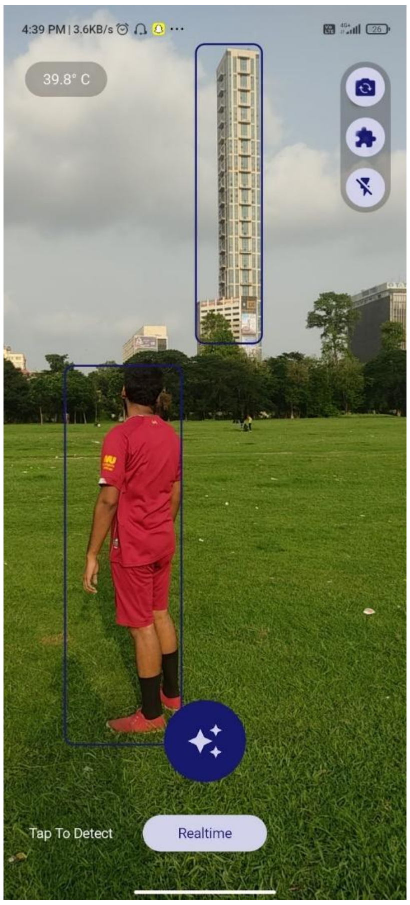


Figure 2

Our object-detector has 3 modes for providing flexibility.

● **Still Image** - Users can opt for "Still Image" mode in the application, where the user can select a single image from any data source, and perform object detection on the image. Refer to Figure 3.

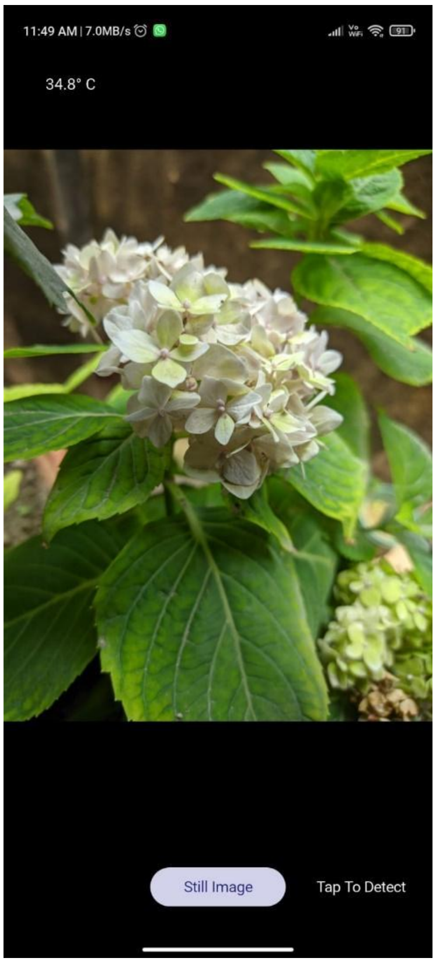


Figure 3

● **Capture Mode** - Using this mode, the user will have to point and capture an image with the button being displayed on the screen. Refer to Figure 4.

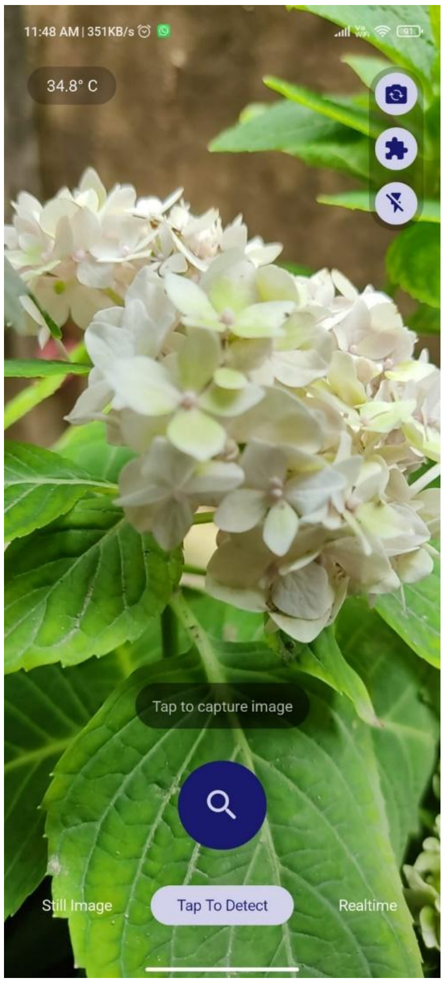


Figure 4

● **Real-Time** - This is the main USP of our application. Users can switch to "Realtime" mode and the application will automatically detect and classify object(s) with the device being placed in front of the object(s). Refer to Figure 5.

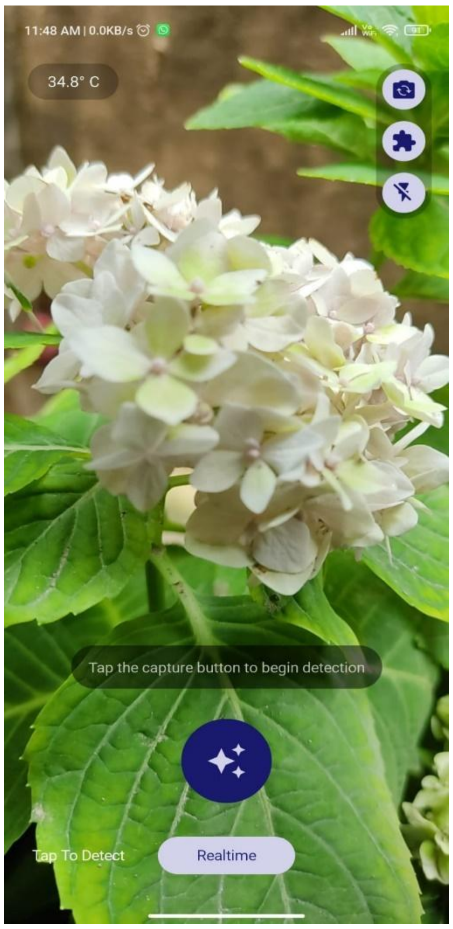


Figure 5

## 3.2 Working Principle

Object detection algorithms can be classified into two main types:

- One stage
- Two stage detection.

In general, deep learning-based detectors extract features from the input image or video frame. An object detector solves two subsequent tasks:

- Find an arbitrary number of objects (possibly even zero), and
- Classify every single object and estimate its size with a bounding box.

Realtime Object detection and classification application is based on a segregated version of two staged object detection. Initially we will use an object detector from ML Kit API which can detect objects with greater accuracy and negligible latency. It can output a bounding box, tracking ID, and a coarse classified label (ML Kit can only classify up to 5 categories: **fashion goods, food, home goods, places, and plants**. Once an object has been assigned with a label and tracking ID, we will pass that data to an external classifier, depending on the label e.g, if the label is of FOOD category we pass it to the FOOD classifier model which will give a label. The tracking ID will not change either until a new object has been detected or the detected object is out of frame. Once the object is classified by the external classifier, we can update the label on the bounding box. This operation will be performed asynchronously in a non-blocking fashion.

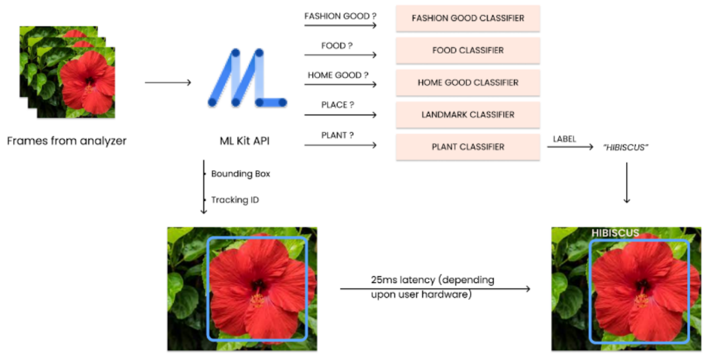


Figure 6

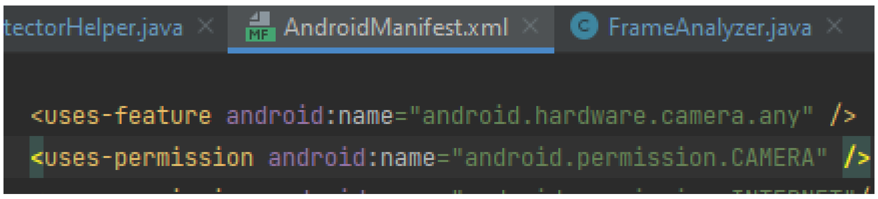
However this will result in a lag between object detection and classifying the detected object, which will completely depend on the hardware spec of the user.

The working principle can be broken down into three steps :
i) Getting frames using CameraX API.
ii) Passing the frames to ML Kit API.
iii) The output label received from ML Kit API is then transferred to an External classifier , where classification of objects is being done at a narrow level.

**i) Generating camera frames using CameraX API :**

CameraX is a Jetpack library, built to help make camera app development easier. It's recommended by Google to work with CameraX. It provides a consistent, easy-to-use API that works across the vast majority of Android devices, with backward-compatibility to Android 5.0 (API level 21).

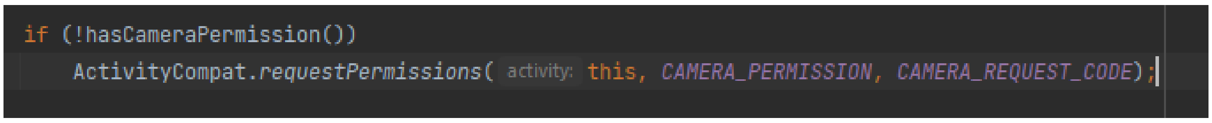
In the initial steps, we have provided camera permissions. This has been done by declaring the required permissions to run a camera in the AndroidManifest.xml file. The AndroidManifest.xml file is the scope which has been created to read all sorts of permission that the application will need for its unhindered run. Figure 7 depicted below shows the actions we have performed to achieve this.



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

Figure 7

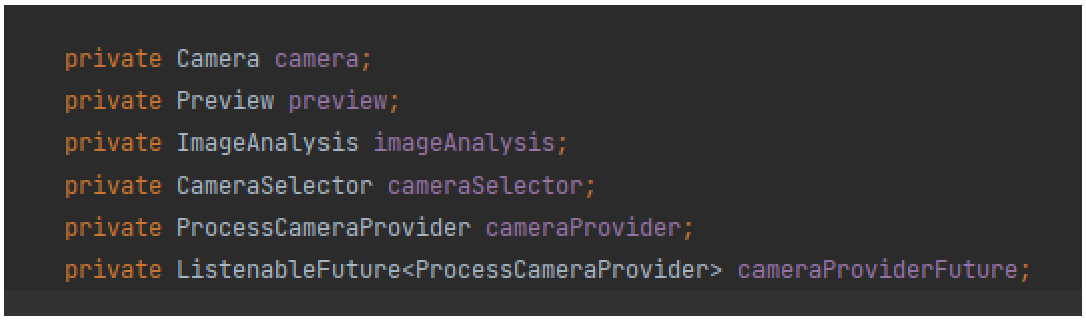
The parameters to request permissions are non-relevant. Here, we have assumed that the user will provide all the required permissions to run the application, and thereby, the application wouldn’t handle the test cases for “permissions-denied” use case. Refer to Figure 8 for this.



```
if (!hasCameraPermission())
    ActivityCompat.requestPermissions( activity: this, CAMERA_PERMISSION, CAMERA_REQUEST_CODE);
```

Figure 8

So, the next steps were to declare the necessary classes and interfaces for initializing the camera. Figure 9 shows the classes and interfaces we have declared for the purpose to be fulfilled.



```
private Camera camera;
private Preview preview;
private ImageAnalysis imageAnalysis;
private CameraSelector cameraSelector;
private ProcessCameraProvider cameraProvider;
private ListenableFuture<ProcessCameraProvider> cameraProviderFuture;
```

Figure 9

- **Camera** - The Camera interface is used to control the flow of data to use cases, control the camera via the CameraControl, and publish the state of the camera via CameraInfo.

- **Preview** - A use case that provides a camera preview stream for displaying on-screen. The Preview stream is connected to the Surface provided via Preview.SurfaceProvider. The application decides how the Surface is shown, and is responsible for managing the Surface lifecycle after providing it.

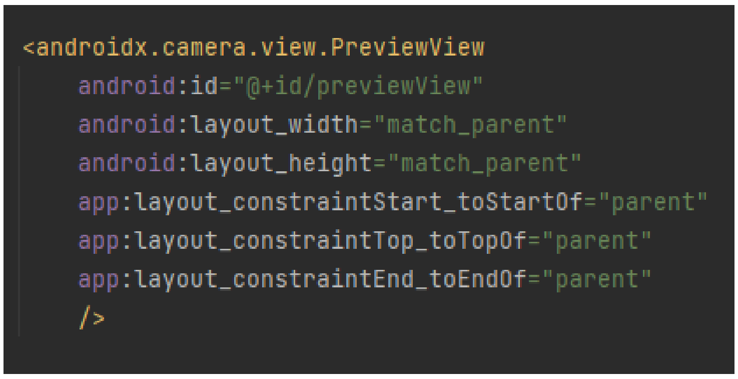
- **ImageAnalysis** - A use case providing CPU accessible images for an app to perform image analysis on. ImageAnalysis acquires images from the camera via an ImageReader. Each image is provided to an ImageAnalysis.Analyzer function which can be implemented by application code, where it can access image data for application analysis via an ImageProxy.

- **CameraSelector** - A set of requirements and priorities used to select a camera or return a filtered set of cameras.

- **ProcessCameraProvider** - A singleton which can be used to bind the lifecycle of cameras to any LifecycleOwner within an application's process.

- **ListenableFuture** - A Future that accepts completion listeners. Each listener has an associated executor, and it is invoked using this executor once the future's computation is complete. If the computation has already completed when the listener is added, the listener will execute immediately.

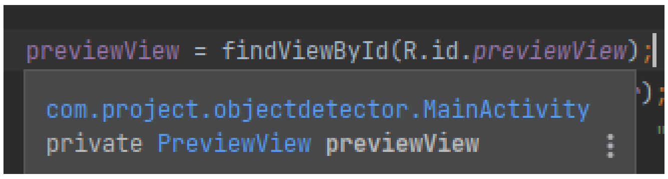
The next step was to declare a Preview view in activity_main.xml file. Preview View is a Custom View that displays the camera feed for CameraX's Preview use case.
This class manages the preview Surface's lifecycle. It internally uses either a TextureView or SurfaceView to display the camera feed, and applies required transformations on them to correctly display the preview, this involves correcting their aspect ratio, scale and rotation. Refer to Figure 10.



```
<androidx.camera.view.PreviewView
    android:id="@+id/previewView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    />
```

Figure 10

So, after creating a Preview view, the next part was to get an instance of the Preview view by the id provided. Refer to Figure 11.

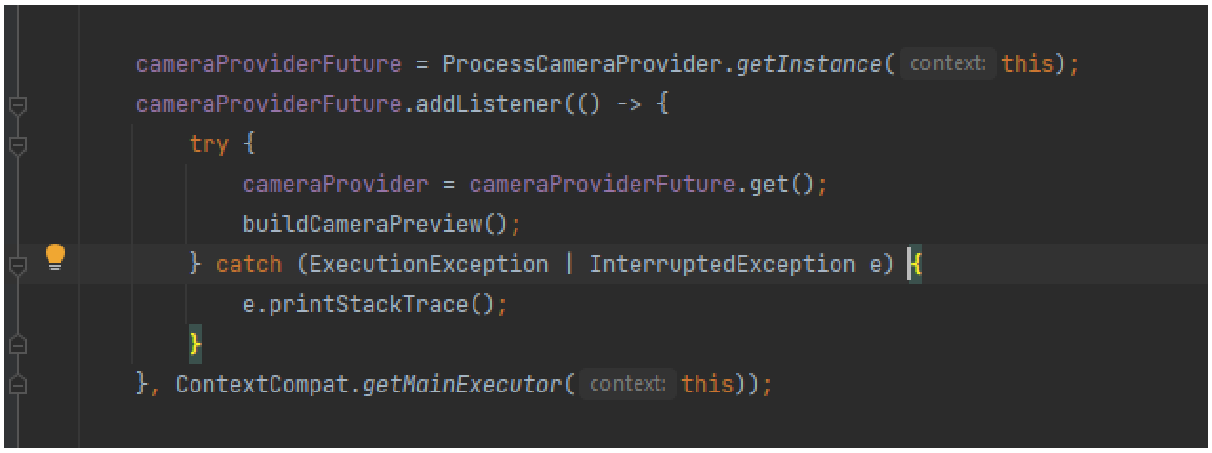


```
previewView = findViewById(R.id.previewView);
  com.project.objectdetector.MainActivity
  private PreviewView previewView
```

Figure 11

cameraProviderFuture takes a ListenableFuture which can be obtained from ProcessCameraProvider.getInstance(). getInstance() retrieves the ProcessCameraProvider associated with the current process.The instance returned here can be used to bind use cases to any LifecycleOwner with bindToLifecycle (LifecycleOwner, CameraSelector, UseCase...). The instance's configuration may be customized by subclassing the application's Application class and implementing CameraXConfig.Provider. For example, the following will initialize this process camera provider with a Camera2 implementation from Camera2 API, and with a custom executor. Refer to Figure 12.

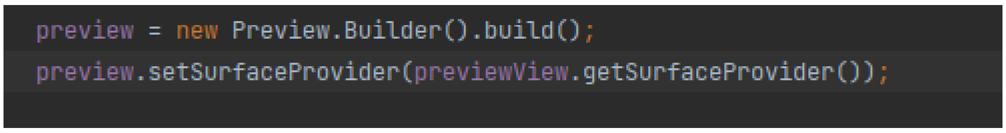
Once the ProcessCameraProvider gets the camera access, we can initialize cameraProvider object with cameraProviderFuture.get() in an runnable
Inside which we call buildCameraPreview method() for initializing the camera. Refer to Figure 12.



```java
cameraProviderFuture = ProcessCameraProvider.getInstance( context: this);
cameraProviderFuture.addListener(() -> {
    try {
        cameraProvider = cameraProviderFuture.get();
        buildCameraPreview();
    } catch (ExecutionException | InterruptedException e) {
        e.printStackTrace();
    }
}, ContextCompat.getMainExecutor( context: this));
```

Figure 12

Next task was to specify the PreviewSurface we will be getting from Preview view. Setting the surfaceProvider will signal to the camera that the use case is ready to receive data. The provider will be triggered on the main thread. Refer to Figure 13.

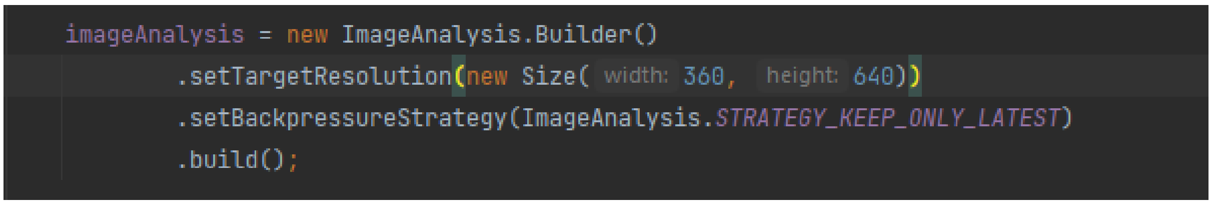


```java
preview = new Preview.Builder().build();
preview.setSurfaceProvider(previewView.getSurfaceProvider());
```

Figure 13

In imageAnalysis, we had specified the resolution of the image we want to process with MLKit API. Generally, it's a good practice to specify an image of lower resolution, but providing a lower size could result in inaccuracy.

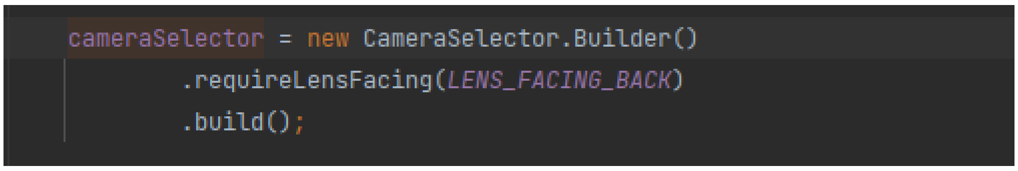
Assigning the backpressure strategy to STRATEGY_KEEP_ONLY_LATEST guarantees only one image will be delivered for analysis at a time. If more images are produced when the analyzer is busy, they will be dropped automatically and not queued for delivery. Once the image being analysed is closed by calling ImageProxy.close(), the next latest image will be delivered. Refer to Figure 14.



```
imageAnalysis = new ImageAnalysis.Builder()
        .setTargetResolution(new Size( width: 360,  height: 640))
        .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
        .build();
```

Figure 14

We have also provided the option to flip the camera, either the rear camera will be enabled or the front camera. By default, the rear camera will be enabled when the user launches the application. The cameraSelector instance will select from either front or rear camera and then call buildCamera(). Refer to Figure 15.

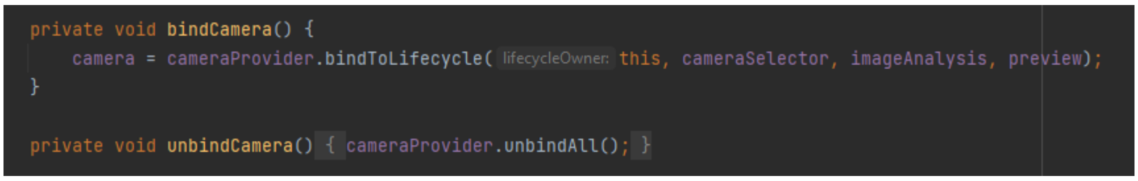


```
cameraSelector = new CameraSelector.Builder()
        .requireLensFacing(LENS_FACING_BACK)
        .build();
```

Figure 15

While defining the buildCamera() method, we pass the above specified objects as parameters into the cameraProvider.bindToLifecycle method. It makes the camera lifecycle aware.cameraProvider.bindToLifecycle binds the camera to lifecycle events (onPause , onResume, onDestroyed)

Similarly unbindCamera() will detach the listener and also close the camera feed (we need to bind and unbind camera for switching to still image mode which doesn't need the camera to be open). Refer to Figure 16.

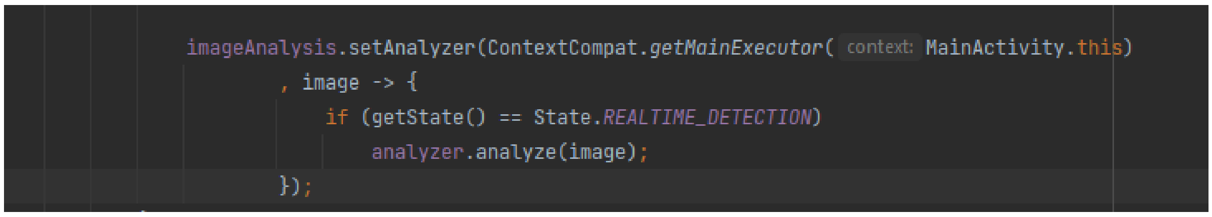


```java
private void bindCamera() {
    camera = cameraProvider.bindToLifecycle( lifecycleOwner: this, cameraSelector, imageAnalysis, preview);
}

private void unbindCamera() { cameraProvider.unbindAll(); }
```

Figure 16

Once the camera is bound, the camera preview will be live on the previewView.

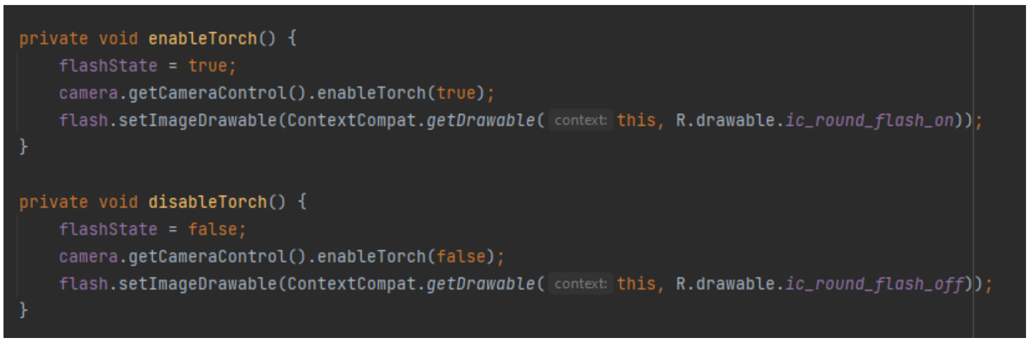
The following code will send a signal to the camera to start generating (360*640) images. It takes a custom Executor, and gives an image object which we can pass into the analyzer class and then to the MLKit API. Refer to Figure 17.



```java
imageAnalysis.setAnalyzer(ContextCompat.getMainExecutor( context: MainActivity.this)
        , image -> {
            if (getState() == State.REALTIME_DETECTION)
                analyzer.analyze(image);
        });
```

Figure 17

Next, we have provided the functionalities for turning on/off the flashlight of the smartphone. Visibility is the key to detect objects with greater accuracy. Refer to Figure 18.



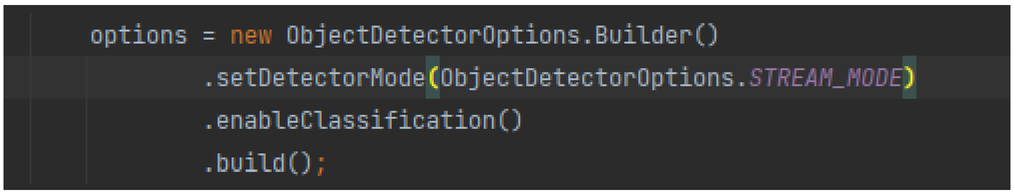
```java
private void enableTorch() {
    flashState = true;
    camera.getCameraControl().enableTorch(true);
    flash.setImageDrawable(ContextCompat.getDrawable( context: this, R.drawable.ic_round_flash_on));
}

private void disableTorch() {
    flashState = false;
    camera.getCameraControl().enableTorch(false);
    flash.setImageDrawable(ContextCompat.getDrawable( context: this, R.drawable.ic_round_flash_off));
}
```

Figure 18

**ii) Passing the frames to ML Kit API :**

When you pass an image to ML Kit, it detects up to five objects in the image along with the position of each object in the image. When detecting objects in video streams, each object has a unique ID that you can use to track the object from frame to frame. You can also optionally enable coarse object classification, which labels objects with broad category descriptions.

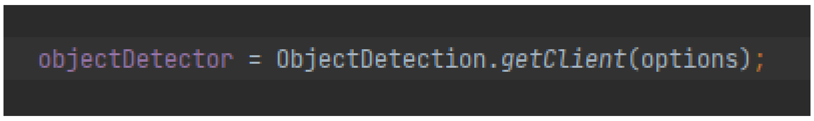
To detect and track objects, first we create an instance of ObjectDetector and optionally specify any detector settings that you want to change from the default. Configure the object detector for your use case with an ObjectDetectorOptions object. Refer to Figure 19.



```
options = new ObjectDetectorOptions.Builder()
        .setDetectorMode(ObjectDetectorOptions.STREAM_MODE)
        .enableClassification()
        .build();
```

Figure 19

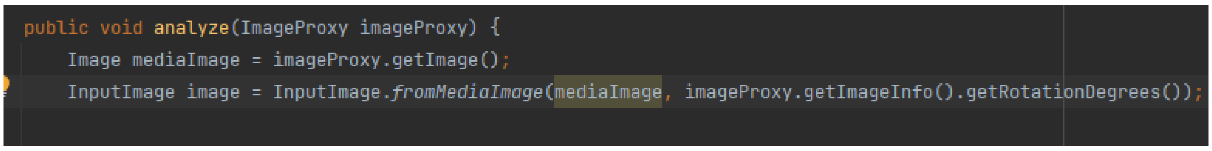
Then we get an instance of ObjectDetector. Refer to Figure 20.



```
objectDetector = ObjectDetection.getClient(options);
```

Figure 20

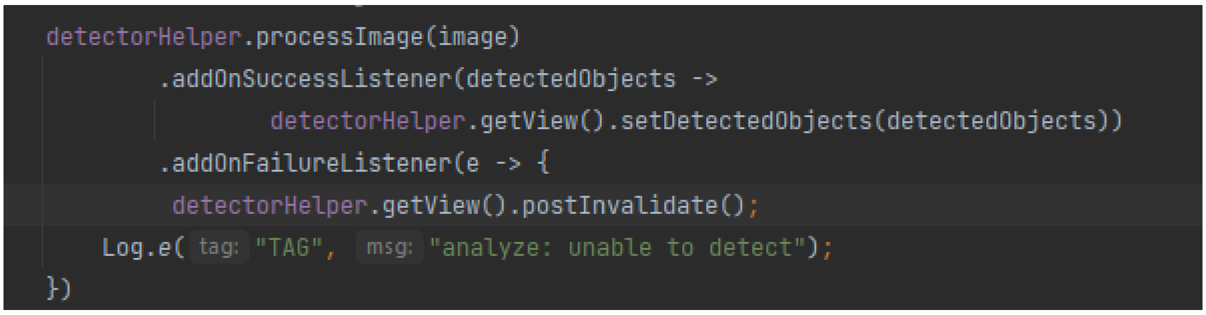
To detect and track objects, pass images to the ObjectDetector instance's process() method. The object detector runs directly from a Bitmap, NV21 ByteBuffer or a YUV_420_888 media.Image. Constructing an InputImage from those sources is recommended if we have direct access to one of them. If we construct an InputImage from other sources, we will handle the conversion internally for us and it might be less efficient.To create an InputImage object from a media.Image object, such as when you capture an image from a device's camera, pass the media.Image object and the image's rotation to InputImage.fromMediaImage(). Refer to Figure 21.



```
public void analyze(ImageProxy imageProxy) {
    Image mediaImage = imageProxy.getImage();
    InputImage image = InputImage.fromMediaImage(mediaImage, imageProxy.getImageInfo().getRotationDegrees());
```

Figure 21

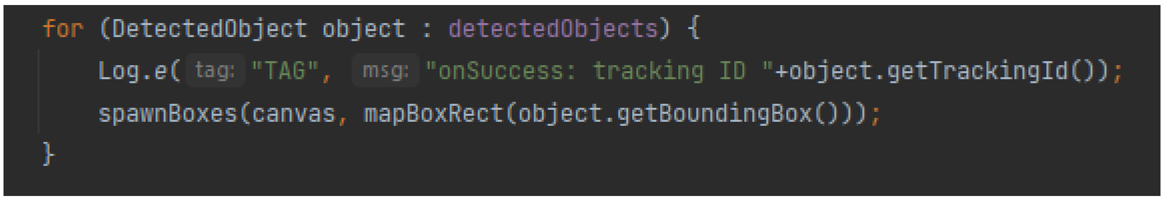
To process the image we can pass the image to the process() method. If the call to process() succeeds, a list of DetectedObjects is passed to the success listener. Refer to Figure 22.



```
detectorHelper.processImage(image)
        .addOnSuccessListener(detectedObjects ->
                detectorHelper.getView().setDetectedObjects(detectedObjects))
        .addOnFailureListener(e -> {
        detectorHelper.getView().postInvalidate();
    Log.e( tag: "TAG",  msg: "analyze: unable to detect");
})
```

Figure 22

Each DetectedObject contains a Bounding box, Tracking ID, Labels. We will draw the bounding box as soon as the object is detected. Refer to Figure 23.



```
for (DetectedObject object : detectedObjects) {
    Log.e( tag: "TAG",  msg: "onSuccess: tracking ID "+object.getTrackingId());
    spawnBoxes(canvas, mapBoxRect(object.getBoundingBox()));
}
```

Figure 23

iii) **The output label received from ML Kit API is then transferred to an External classifier , where classification of objects is being done at a narrow level :**

The MLKit API's default object detector classifies objects into the following categories: fashion goods, food, home goods, places, and plants. We pass the result from the object detector to the specific classifier. This classifier will detect the object from the given bounding box and return a label. Then the label can be placed over the bounding box.

The tracking ID will be used to track the object in successive frames, if the object is lost (camera is moved or a new object is assigned that tracking ID) then we will revoke the classification process and shut down the external classifier immediately.
This two-step process will result in additional latency, which could be significantly high in smartphones with poor chipset. But once the object is classified the label will only be updated if the tracking ID changes on the object. This external classification step is performed in a separate executor asynchronously.

## 3.3 USP

As of now, you might have come to know that real-time object detection is a very demanding task on its own. Modern computers can run object detection models without a break, with latencies in nanoseconds. Although we have come across a long way technologically we still struggle with real-time object detection on mobile phones. But nowadays a large proportion of mobile phones have decent processing power, and with the latest advancements now we can detect objects in mobile phones with lower latencies in single digits, we can even detect multiple objects with decent accuracy. But it comes with a caveat, while we can detect a wide range of objects, it can only classify them coarsely.

Since the applications of real-time object detection are overwhelmingly increasing we decided to come up with a prototype of our own, demonstrating the fundamentals of object detection and a workaround for classification in mobile devices.
So, here comes our vital aim, that is to deal with real-time object detection which makes it easier for users to meet their needs in terms of object-detection, and thereby, providing a seamless user experience. Just, turn the application ON and toggle through real-time mode. Click on the button mentioned there. The application will now start automatically detecting objects which will be captured through camera lens.

The images below illustrate the stages the application can perform while being in real-time mode.

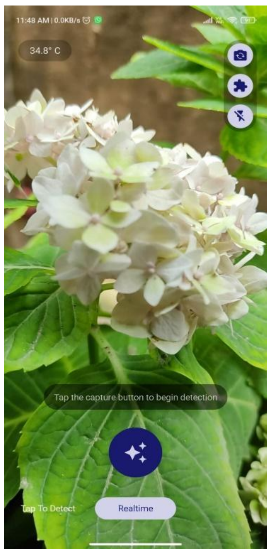


Figure 24

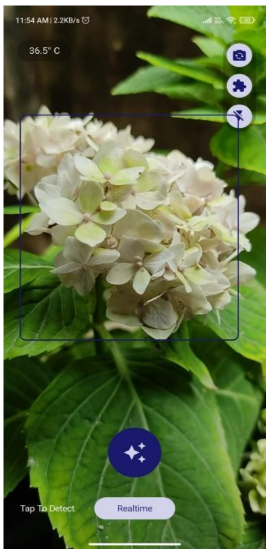


Figure 25

## 3.4 Libraries

### 3.4.1 CameraX

CameraX is a Jetpack support library, built to help you make camera app development easier. It provides a consistent and easy-to-use API surface that works across most Android devices, with backward-compatibility to Android 5.0 (API level 21).

**Why did we opt for CameraX ?**

- **Ease of use -** CameraX emphasizes use cases, which allow you to focus on the task you need to get done instead of managing device-specific nuances. Most common camera use cases are supported:

Preview: View an image on the display.
Image analysis: Access a buffer seamlessly for use in your algorithms, such as to pass to ML Kit.

- **Consistency across devices -** Maintaining consistent camera behavior is hard. We have to consider aspect ratio, orientation, rotation, preview size, and image size. With CameraX, these basic behaviors work properly.

### 3.4.2 RxJava and RxAndroid

- RxJava is a Java VM implementation of Reactive Extensions: a library for composing asynchronous and event-based programs by using observable sequences.

It extends the observer pattern to support sequences of data/events and adds operators that allow you to compose sequences together declaratively while abstracting away concerns about things like low-level threading, synchronization, thread-safety and concurrent data structures.

- This module adds the minimum classes to RxJava that make writing reactive components in Android applications easy and hassle-free. More specifically, it provides a Scheduler that schedules on the main thread or any given Looper.

RxJava operators don't work with Threads or ExecutorServices directly but with so-called Schedulers that abstract away sources of concurrency behind a uniform API. RxJava 3 features several standard schedulers accessible via Schedulers utility class.

Schedulers.computation(): Run computation intensive work on a fixed number of dedicated threads in the background. Most asynchronous operators use this as their default Scheduler.
Schedulers.io(): Run I/O-like or blocking operations on a dynamically changing set of threads.
Schedulers.single(): Run work on a single thread in a sequential and FIFO manner.
Schedulers.trampoline(): Run work in a sequential and FIFO manner in one of the participating threads, usually for testing purposes.

These are available on all JVM platforms but some specific platforms, such as Android, have their own typical Schedulers defined: AndroidSchedulers.mainThread(), SwingScheduler.instance() or JavaFXSchedulers.gui().

### 3.4.3 MLKit

With ML Kit's on-device Object Detection and Tracking API, we can detect and track objects in an image or live camera feed.
Optionally, we can classify detected objects, either by using the coarse classifier built into the API, or using your own custom image classification model. See Using a custom TensorFlow Lite model for more information.
Because object detection and tracking happens on the device, it works well as the front end of the visual search pipeline. After you detect and filter objects, you can pass them to a cloud backend, such as Cloud Vision Product Search.

**Why did we opt for MLKit ?**

- Fast object detection and tracking - We can detect objects and get their locations in the image. Also, we can track objects across successive image frames.
- Optimized on-device model - The object detection and tracking model is optimized for mobile devices and intended for use in real-time applications, even on lower-end devices.
- Prominent object detection - It can automatically determine the most prominent object in an image.
- Coarse classification - It can classify objects into broad categories, which we can use to filter out objects we aren't interested in.
- 5. Classification with a custom model - We can use our own custom image classification model to identify or filter specific object categories. We can even make our custom model perform better by leaving out the background of the image.

## 3.5 Tools Used

1. **Github** - GitHub is a popular programming resource used for code sharing. It's a social networking site for programmers that many companies and organizations use to facilitate project management and collaboration. Git is used to coordinate the workflow among project team members and track their progress over time. It also benefits both programmers and non-technical users by keeping track of their project files. Git allows multiple users to work together without disrupting each other's work. The version control system, or VCS, is the element in Git that is best suited for tackling Decathlon's two problems. The Git version control system, as the name suggests, is a system that records all the modifications made to a file or set of data so that a specific version may be called up later if needed. The system makes sure that all the team members are working on the file's latest version, and everyone can work simultaneously on the same project.

Hence, we used Github so that everything remains organized till the completion of our project. Moderation tools like Issue and Pull Request Locking helped our team to discuss any implementations before changing the source code.

2. **Figma -** Figma is a cloud-based design tool that is similar to Sketch in functionality and features, but with big differences that make Figma better for team collaboration. Figma simplifies the design process and is more effective than other programs at helping designers and teams work together efficiently.

Another big advantage of Figma is that it allows real-time collaboration on the same file. When using conventional "offline" apps like Sketch and Photoshop, if any one of us want to share their work, they typically have to export it to an image file, then send it via email or instant message.
In Figma, instead of exporting static images, we can simply share a link to the Figma file for group members to open in their browser. This in itself saved significant time and inconvenience in our workflow. But more importantly, it means that we can interact more richly with the work, and review the latest version of the file.

Hence, we used Figma to design the prototype of the User-Interface of our application in initial stages.

Below mentioned images show the stages of creating our User Interface's prototype.

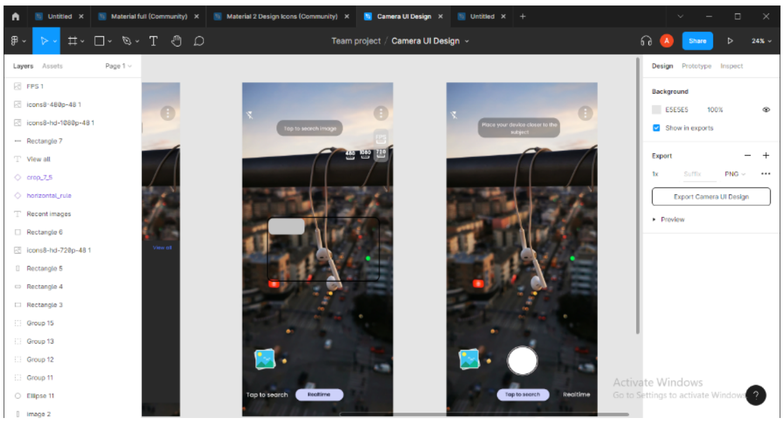


Figure 26

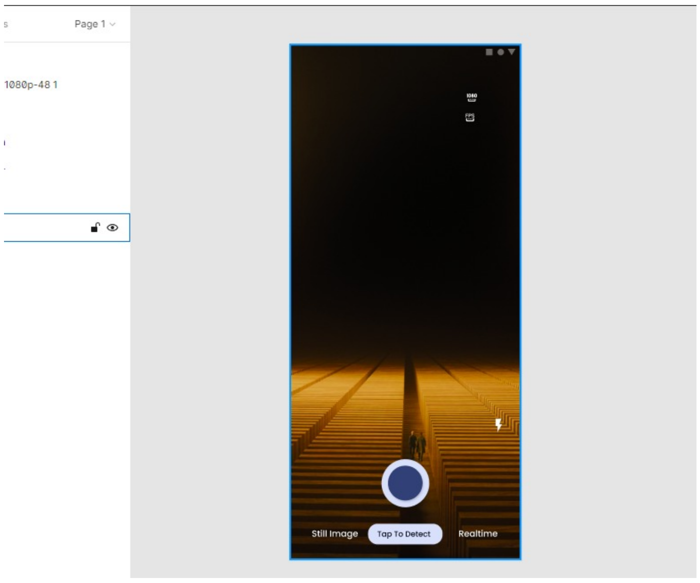


Figure 27

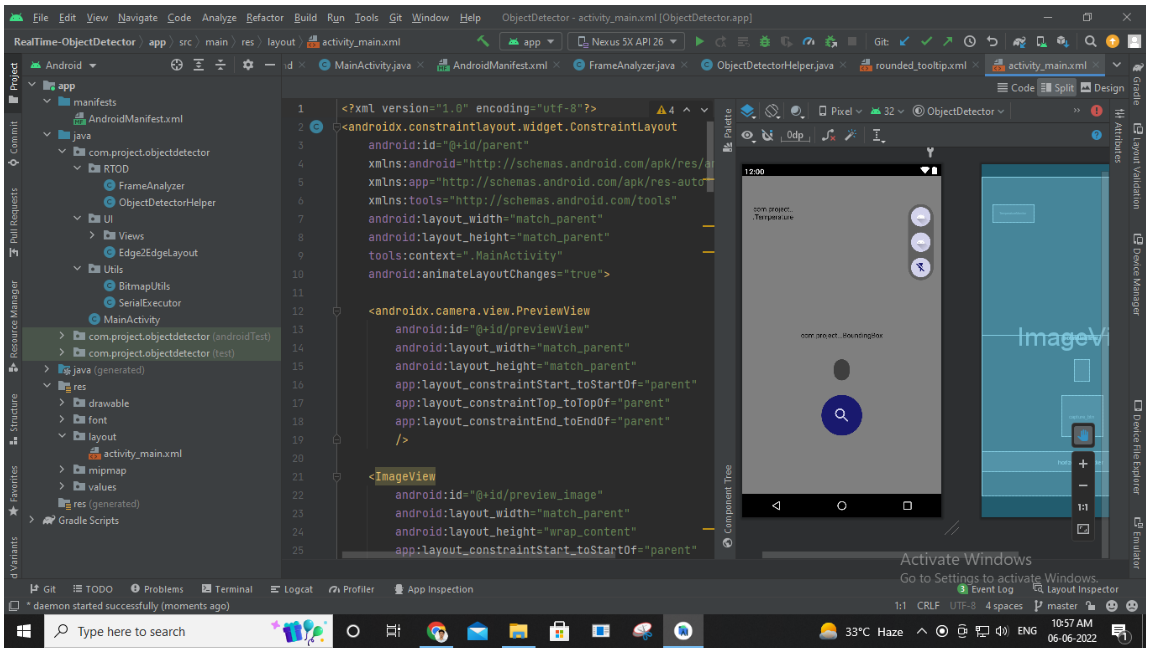
3. **Android Studio -** Android Studio is the official IDE for Android application development, based on IntelliJ IDEA. By default, Android Studio displays the project files in the Android project view. This view shows a flattened version of the project's structure that provides quick access to the key source files of Android projects and helps you work with the Gradle-based build system.

Android Studio offers:

- Flexible Gradle-based build system
- Build variants and multiple apk file generation
- Code templates to help us build common app features
- Rich layout editor with support for drag and drop theme editing
- Lint tools to catch performance, usability, version compatibility, and other problems
- ProGuard and app-signing capabilities
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine

Thus, this was the best platform we could have now to build our project due to the above mentioned capabilities of the IDE.

Figure 28 shows the stage of creating the User Interface on Android Studio.



```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    android:id="@+id/parent"
    xmlns:android="http://schemas.android.com/apk/res/a
    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"
    android:animateLayoutChanges="true">

    <androidx.camera.view.PreviewView
        android:id="@+id/previewView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        />

    <ImageView
        android:id="@+id/preview_image"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintStart_toStartOf="parent"
```

Figure 28

## 3.6 Limitations

The field being quite new and the applications of real-time object detection are overwhelmingly increasing, most of the strategies and techniques we used have limitations. Some of them are mentioned below.

- There’s a delay in time between object’s detection and its classification. This is because we are using an external classifier for identification of the objects. So, depending upon the model we are using, and the device’s hardware configurations, there will be an additional increase in latency.

- Due to the increase in latency, the device(if its a low end device) where the application is running might undergo overheating or dropping in frames

- Since our application is not storing user’s data in cloud or server, the overall application is consuming large memory for storage in the device itself.

## 3.7 Future Scope

Since, object-detection is a very vast field. We can include some additional features in our application so that it can be accessible to most kinds of users.

- We can implement computer vision to assist people with low vision or blindness to get things done faster and more easily. By implementing the “Talkback screen reader” feature here, to describe the object for someone with little or no sight.

- The talkback feature will also read out texts in the image. If you want to read everything present in the image, one can switch to “Document” mode.

- We can implement text detection in our application. The prompt will read out the texts loud using the talkback feature. “Translation” options will also be provided along with “Languages” as option to change as per user’s convenience.

# CHAPTER 4

# RESULTS

## 4.1 Discussion

Detection of images or moving objects have been highly worked upon, and has been integrated and used in commercial, residential and industrial environments. But, most of the strategies and techniques have heavy limitations. One of the limitations is due to low computational resources at user level. Other important limitations that need to be tackled are lack of proper data analysis of the measured trained data, dependency on the motion of the objects, inability to differentiate one object from other, and also concern over speed of the object under detection and illuminance.

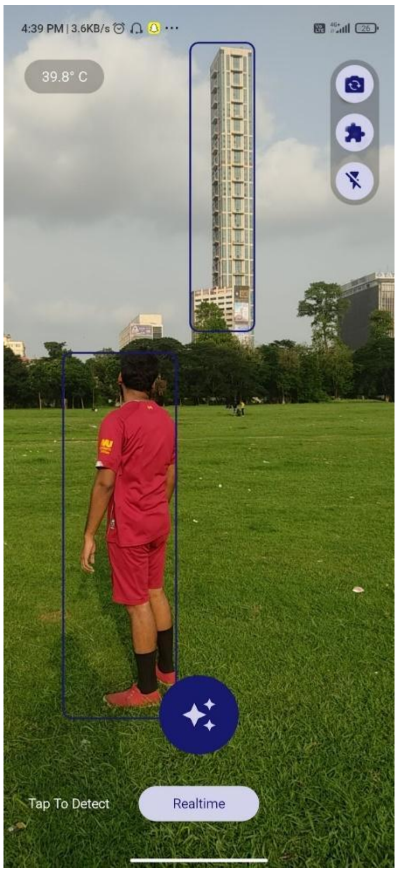


Figure 29

( Before Detection )

This is a sample image we feed to the algorithm and expect our algorithm to detect and identify objects in the image and label them according to the class assigned to it.

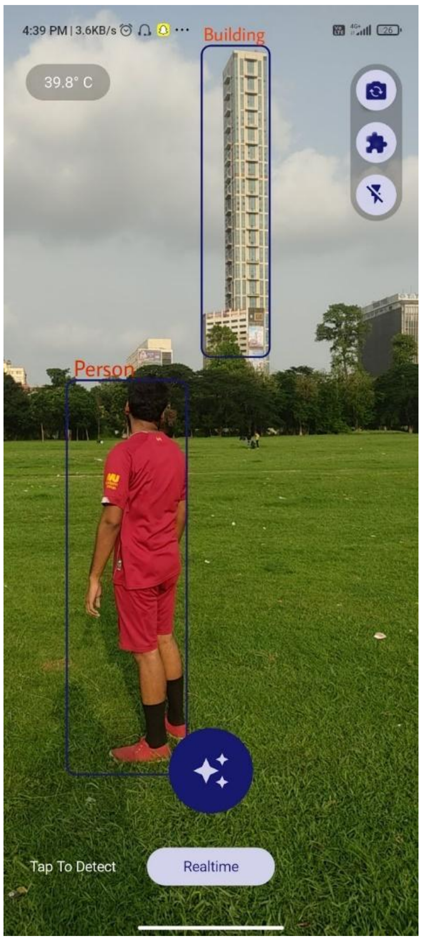


Figure 30

( After Detection )

As expected, our algorithm identifies the objects by its classes and classifies each object.
This application supports multi-object detection, which is very useful. All the steps occur simultaneously with great speeds giving remarkable results, detecting all the categories of the trained model within a good illuminance.

## 4.2 Conclusion

By using this application and based on experimental results, we are able to detect objects more precisely and identify the objects individually. This report also provides experimental results on different methods for object detection and identification and compares each method for their efficiencies. The objects are limited to identify only a limited number of categories but the scope can be broadened with a revised trained model.

# APPENDIX

Listed below are various extensions to references in the text. They are here to provide the reader with extra detail that may be required but was not present in the main report chapters.

1. Application Programming Interfaces :

1.1 CameraX

CameraX is a Jetpack library, built to help make camera app development easier. For new apps, we recommend starting with CameraX. It provides a consistent, easy-to-use API that works across the vast majority of Android devices, with backward-compatibility to Android 5.0 (API level 21).

1.2 ML Kit

ML Kit brings Google's machine learning expertise to mobile developers in a powerful and easy-to-use package. Make your iOS and Android apps more engaging, personalized, and helpful with solutions that are optimized to run on device. With ML Kit's on-device Object Detection and Tracking API, you can detect and track objects in an image or live camera feed. Optionally, you can classify detected objects, either by using the coarse classifier built into the API, or using your own custom image classification model.

2. Libraries

2.1 RxJava

RxJava is a Java VM implementation of Reactive Extensions: a library for composing asynchronous and event-based programs by using observable sequences.It extends the observer pattern to support sequences of data/events and adds operators that allow you to compose sequences together declaratively while abstracting away concerns about things like low-level threading, synchronization, thread-safety and concurrent data structures.

2.2 RxAndroid

This module adds the minimum classes to RxJava that make writing reactive components in Android applications easy and hassle-free. More specifically, it provides a Scheduler that schedules on the main thread or any given Looper.

## 3. Definitions

### 3.1 Image processing:

Image processing is a method to perform operations on an image , in order to get an enhanced image or to extract useful information from it.

### 3.2 Object localization:

Object localization refers to identifying the location of one or more objects in an image and drawing a bounding-box around their extent.

### 3.3 Computer Vision:

Computer vision is a field of artificial intelligence that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs.

### 3.4. Deep Learning:

Deep learning is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised.

# REFERENCES

1. Erhan, D., Szegedy, C., Toshev, A., and Anguelov, D. (2014). “Scalable object detection using deep neural networks,” in Computer Vision and Pattern Recognition Frontiers in Robotics and AI www.frontiersin.org November 2015.

2. Agarwal, S., Awan, A., and Roth, D. (2004). Learning to detect objects in images via a sparse,part-based representation. IEEE Trans. Pattern Anal. Mach. Intell. 26,1475–1490.doi:10.1109/TPAMI.2004.108

3. Alexe, B., Deselaers, T., and Ferrari, V. (2010). “What is an object?,” in ComputerVision and PatternRecognition (CVPR), 2010 IEEE Conference on (San Francisco,CA: IEEE), 73–80.doi:10.1109/CVPR.2010.5540226

4. Andreopoulos, A., and Tsotsos, J. K. (2013). 50 years of object recognition: direc-tions forward.Comput. Vis. Image Underst. 117, 827–891. doi:10.1016/j.cviu.2013.04.005

5. Azizpour, H., and Laptev, I. (2012). “Object detection using strongly-supervised deformable part models,” in Computer Vision-ECCV 2012 (Florence: Springer),836–849.

6. Cadena, C., Dick, A., and Reid, I. (2015). “A fast, modular scene understanding sys-tem using context-aware object detection,” in Robotics and Automation (ICRA),2015 IEEE International Conference on (Seattle, WA).

7. Correa, M., Hermosilla, G., Verschae, R., and Ruiz-del-Solar, J. (2012). Human Detection and identification by robots using thermal and visual information in domestic environments. J. Intell. Robot Syst. 66, 223–243. doi:10.1007/s10846-011-9612-2

8. https://viso.ai/deep-learning/object-detection/

9. https://developers.google.com/ml-kit/vision/object-detection

10. https://developer.android.com/training/camerax