

## **Google AI-ML Virtual Internship**

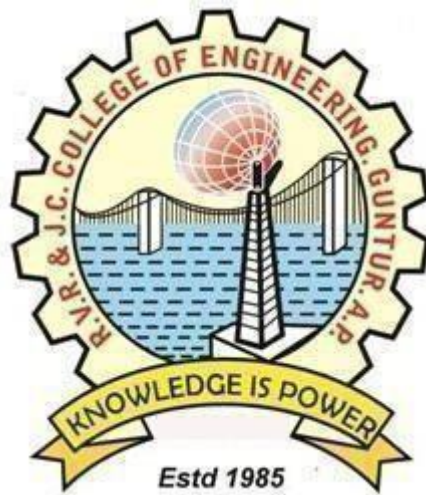
Submitted in partial fulfillment of requirements to CSE (Data Science)

**Summer Internship (CD - 451)**

**IV/IV B. Tech CSE(DS) (VII Semester )**

Submitted by

**LAKSHMI PRASANNA VALANUKONDA(Y22CD172)**



AUGUST 2025

R.V.R. & J.C. College of Engineering (Autonomous)

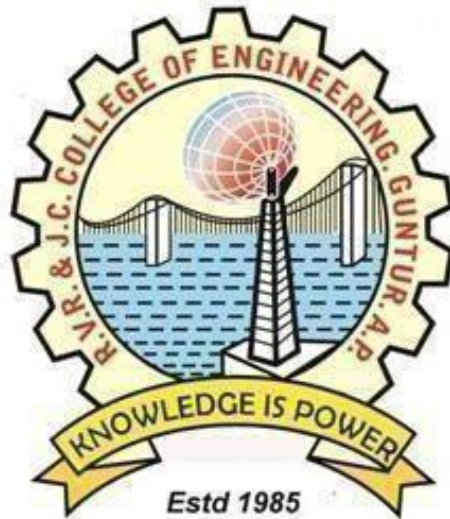
(NAAC A+ Grade) (Approved by A.I.C.T.E.)

(Affiliated to Acharya Nagarjuna University)

Chandramoulipuram::Chowdavaram

Guntur – 522019

**R. V. R. & J.C. COLLEGE OF ENGINEERING**  
DEPARTMENT OF  
COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)



**CERTIFICATE**

This is to certify that this internship report “**Google AI-ML Virtual Internship**” is the Bonafide work of “**LAKSHMI PRASANNA VALANUKONDA(Y22CD172)**” who has carried out the work under my supervision and submitted in partial fulfillment for the award of **Summer Internship (CD-451)** during the year 2025 - 2026.

**Dr. Ganji Ramanjaiah**  
Internship Incharge

**Dr. M. V. P.Chandra Sekhara Rao**  
Prof. & HEAD OF DEPARTMENT,CSE(DS)

## ACKNOWLEDGEMENT

I would like to express our sincere gratitude to these dignitaries, who are with us in the journey of my Summer Internship “**Google AI-ML Virtual Internship.**”

First and foremost, we extend our heartfelt thanks to **Dr. Kolla Srinivas**, principal of **R. V. R. & J.C. College of Engineering**, Guntur, for providing me with such an overwhelming environment to undertake this internship.

I am utmost grateful to **Dr. M. V. P. Chandra Shekara Rao**, Head of the Department of Computer Science and Engineering (Data Science) for paving a path for me and assisting me to meet the requirements that are needed to complete this internship.

I extend my gratitude to the Incharge **Dr Ganji Ramanjaiah** for her ecstatic guidance and feedback throughout the internship. Her constant support and constructive criticism have helped me in completing the internship.

I would also like to express our sincere thanks to my friends and family for their moral support throughout our journey.

**LAKSHMI PRASANNA VALANUKONDA**

**(Y22CD172)**



# TABLE OF CONTENTS

Title	Page no
<b>Chapter 1 Introduction</b>	
1.1 Description of the Company	1
1.2 Overview of the Project	3
1.3 Technology Stack	4
<b>Chapter 2 Summary of Experience</b>	7
<b>Chapter 3 Reflection on Learning</b>	12
<b>Conclusion</b>	40

## List Of Figures

Title	Pg,No
Fig-2.1 Flow of Machine learning	7
Fig 2.2 Machine learning pipe line	8
Fig 2.3 Applications of Computer Vision:	9
Fig 2.4 : Recognizing food & state	11
Fig 2.5 Content recognition in Video Ananlysis	12
Fig-3.1 Product Image Search	16
Fig-3.2 Further Classification of Product Image Search	25
Fig- 3.3 Sample images for the object detection	28
Fig- 3.4 Detection of objects for input image	31
Fig 3.5 Bit Map for the object	33
Fig- 3.6 further detection of object for the input image	40
Fig -3.7 Sample for the image classification	41
Fig – 3.8 Performance metrics for the image classification	47

## List Of Tables

Title	Pg,No
Table 1.1 Overview of the project	3
Table 1.2 Table 1.2 - Components and their description	6

## **Abbreviations**

Here are the abbreviations used in the AI/ML internship:

1. AI - Artificial Intelligence
2. ML - Machine Learning
3. NLP - Natural Language Processing
4. TensorFlow - TF
5. Google Colab - Colab
6. Python
7. Google Developer Profile - GDP
8. Git
9. GitHub
10. Keras
11. OpenCV
12. NLTK (Natural Language Toolkit)
13. TPU (Tensor Processing Unit)
14. GPU (Graphics Processing Unit)

These abbreviations are commonly used throughout the internship to refer to various tools, frameworks, and concepts





# CHAPTER 1 INTRODUCTION

## 1.1 Introduction to Company:

Google, a leading technology company and subsidiary of Alphabet Inc., is synonymous with innovation, excellence, and a culture that fosters creativity and learning. Established in 1998 by Larry Page and Sergey Brin, Google has grown from a search engine startup into a global powerhouse, providing a diverse range of services such as search, cloud computing, advertising, hardware, and software. Google's reputation for pioneering technologies is particularly notable in the fields of artificial intelligence and machine learning, where the company continually pushes the boundaries of what's possible.

Google's AI/ML research has led to groundbreaking advancements, including the development of TensorFlow, an open-source machine learning framework widely used in academia and industry. TensorFlow has become a cornerstone for many AI applications, from image and speech recognition to natural language processing and robotics. Google's influence in AI/ML extends beyond technology to practical applications in healthcare, autonomous vehicles, and more.

The company's mission is to "organize the world's information and make it universally accessible and useful." This mission underpins Google's commitment to creating tools and platforms that improve people's lives and empower them to achieve more. Google's AI/ML internship programs align with this philosophy, providing participants with opportunities to work on real-world projects, collaborate with experienced professionals, and learn from some of the best minds in the industry.

Google's culture is centered around innovation, teamwork, and continuous learning. Interns are encouraged to experiment, take risks, and contribute their unique perspectives. The company fosters an inclusive environment where diversity is valued, and every individual is encouraged to bring their authentic self to work. This emphasis on inclusivity and creativity contributes to Google's status as an attractive workplace for aspiring AI/ML professionals. In addition to its technological achievements, Google is committed to social responsibility and sustainability. The company invests in various initiatives to promote education, environmental sustainability, and

diversity. Google's efforts to reduce its carbon footprint and support renewable energy reflect its broader goal of creating a positive impact on society.

Internships at Google offer a comprehensive experience, combining technical challenges with opportunities for personal growth and professional development. Mentorship is a core component, providing interns with guidance and support as they navigate complex projects and learn industry best practices. Collaborative projects encourage teamwork, while structured code reviews and presentations help interns refine their skills and build confidence.

Google's AI/ML internship is more than a learning opportunity; it's an invitation to join a community of innovators committed to advancing technology and making a meaningful difference in the world. Through this program, interns gain not only technical expertise but also an understanding of Google's culture and values, setting the stage for a successful career in the technology industry. Google's 10 biggest AI moments so far:

1. DeepMind's AlphaGo Victory
2. Launch of TensorFlow
3. BERT: Breakthrough in Natural Language Processing (NLP)
4. Google Assistant
5. Google Photos' AI Features
6. Project Maven
7. AutoML
8. Google Duplex
10. AI in Google Health Initiatives

Aspect	Description
Program Focus	In-depth learning experience in AI and machine learning, with a focus on computer vision using TensorFlow.

Learning Modules	A series of structured modules that cover key AI/ML concepts and applications.
Hands-On Projects	Practical assignments and projects involving real-world AI/ML tasks, such as object detection and image classification.
Technology Stack	Google's TensorFlow framework, Google Colab for code implementation, and Google Developer Profile for tracking achievements.
Achievements	Earning six Google Developer Profile badges, indicating proficiency in object detection, image classification, and product image search.
Mentorship	Guidance from experienced professionals in AI/ML, with regular feedback and support throughout the internship.
Collaboration	Team-based projects and collaborative learning encourage teamwork and knowledge sharing among interns.
Code Reviews and Presentations	Structured code reviews and project presentations to refine skills and foster effective communication.
Career Development	Opportunities to network with industry experts and gain insights into AI/ML career paths.
Outcome	A comprehensive learning experience that equips interns with practical AI/ML skills and prepares them for future opportunities in the field.

## **1.2 Technology stack**

Technology Stack Used in the AI/ML Internship.

The technology stack for an AI/ML internship at Google is designed to offer comprehensive exposure to industry-standard tools and platforms. This stack enables interns to build, test, and deploy machine learning models while encouraging collaboration and experimentation. Below are the key components of this technology stack:

### **TensorFlow**

TensorFlow is Google's open-source machine learning framework, a critical component of the technology stack. It supports a wide range of machine learning tasks, from simple linear models to complex neural networks. During the internship, interns use TensorFlow to build models for various applications, including image classification, object detection, and natural language processing. The framework provides tools for both low-level computations and high-level abstractions, allowing interns to explore and implement diverse AI concepts.

### **Google Colab**

Google Colab is an online Jupyter notebook environment that facilitates code development and collaboration. It allows interns to write, execute, and share Python code in a cloud-based setting, removing the need for local installations and enabling easy access to powerful hardware resources, such as GPUs and TPUs. Colab is instrumental in the internship, providing an ideal platform for developing, testing, and debugging machine learning models in a collaborative context.

### **Python**

Python is the primary programming language used in the internship. Its simplicity and versatility make it an excellent choice for AI/ML development. Interns use Python for writing TensorFlow scripts, manipulating data, and integrating various libraries for data analysis and visualization.

The wide range of Python-based tools and packages, like NumPy, Pandas, Matplotlib, and Scikit-Learn, adds to its versatility, allowing interns to perform complex computations and visualize results effectively.

Version control is essential for collaborative development, and Git, along with GitHub, is used to manage code repositories and track changes. Interns work with Git to maintain project history, collaborate with other team members, and submit code for review. GitHub serves as a central platform for hosting code repositories, facilitating code reviews, and enabling collaborative development within the internship.

### **Additional Libraries and Tools**

Beyond TensorFlow, Google Colab, and Python, interns may use other specialized libraries and tools depending on their project requirements. Commonly used libraries include:

**Keras:** A high-level neural network API that integrates with TensorFlow, used for building deep learning models with less code.

**OpenCV:** An open-source library for computer vision tasks, often used for image processing and object detection

**Natural Language Toolkit (NLTK):** A library for natural language processing tasks, such as tokenization, parsing, and text analysis.

These components make up the technology stack for the AI/ML internship, providing a solid foundation for interns to develop and refine their machine-learning skills. The stack's flexibility and versatility allow interns to explore various AI/ML applications while promoting collaboration and learning throughout the program.

Component	Description
TensorFlow	Google's framework for building and deploying machine learning and deep learning models.
Google Colab	Cloud-based Jupyter notebook environment for developing and sharing AI/ML code.
Python	The primary language used for AI/ML development, known for its simplicity and flexibility.
Google Developer Profile	Platform for tracking achievements and earning badges for completed AI/ML learning modules.
Git & GitHub	Tools for version control and collaborative code management, crucial for teamwork.
Additional Libraries	Libraries like Keras, OpenCV, and NLTK for deep learning, computer vision, and natural language processing tasks.

Table 1.2 - Components and their description

# CHAPTER 2 SUMMARY OF EXPERIENCE

## AIML FOUNDATIONS

### 2What is Machine Learning?

Machine learning is the scientific study of algorithms and statistical models to perform a task by using inference instead of instructions.

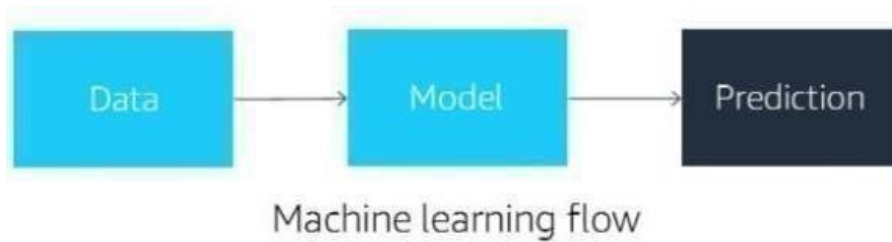


Fig-2.1 Flow of Machine learning

- 2.1 Artificial intelligence is the broad field of building machines to perform human tasks.
- 2.2 Machine learning is a subset of AI. It focuses on using data to train ML models so the models can make predictions.
- 2.3 Deep learning is a technique that was inspired from human biology. It uses layers of neurons to build networks that solve problems.
- 2.4 Advancements in technology, cloud computing, and algorithm development have led to a rise in machine learning capabilities and applications.

**3Business Problems Solved with Machine Learning** Machine learning is used throughout a person's digital life. Here are some examples:

- 3.1 Spam –Your spam filter is the result of an ML program that was trained with examples of spam and regular email messages.
- 3.2 Recommendations –Based on books that you read or products that you buy, ML programs predict other books or products that you might want. Again, the ML program was trained with data from other readers' habits and purchases.
- Credit card fraud –Similarly, the ML program was trained on examples of transactions that turned out to be fraudulent, along with transactions that were legitimate. Machine learning problems can be grouped into –
- Supervised learning: You have training data for which you know the answer.
- Unsupervised learning: You have data, but you are looking for insights within the data.
- Reinforcement learning: The model learns in a way that is based on experience and feedback

- Most business problems are supervised learning.

## 2. Machine Learning Process

The machine learning pipeline process can guide you through the process of training and evaluating a model.

The iterative process can be broken into three broad steps –

- Data processing
- Model training
- Model evaluation ML PIPELINE:

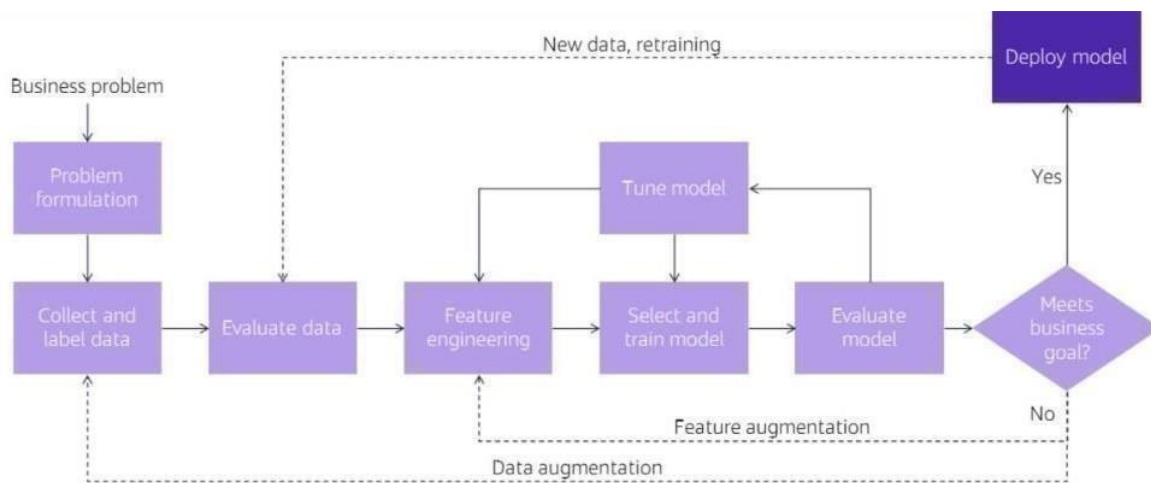


Fig 2.2 Machine learning pipeline

## 2. Machine Learning Tools Overview

- Jupyter Notebook is an open-source web app for creating and sharing live code, equations, visualizations, and text. Jupyter Lab is a web-based interactive development environment for Jupyter notebooks, code, and data. Jupyter Lab is flexible.
- pandas is an open-source Python library. It's used for data handling and analysis. It represents data in a table that is similar to a spreadsheet. This table is known as a panda Dataframe.
- Matplotlib is a library for creating scientific static, animated, and interactive visualizations in Python.
- Seaborn is another data visualization library for Python. It's built on matplotlib, and it provides a high-level interface for drawing informative statistical graphics.



- NumPy, a key Python package for scientific computing, includes functions for N-dimensional arrays and essential math operations like linear algebra, Fourier transform, and random number generation.
- scikit-learn is an open-source ML library for supervised and unsupervised learning, offering tools for model fitting, data pre-processing, selection, evaluation, and more.

## INTRODUCING COMPUTER VISION

**Computer Vision** enables machines to identify people, places, and things in images with accuracy at or above human levels, with greater speed and efficiency. Often built with deep learning models, computer vision automates the extraction, analysis, classification, and understanding of useful information from a single image or a sequence of images. The image data can take many forms, such as single images, video sequences, views from multiple cameras, or three-dimensional data.

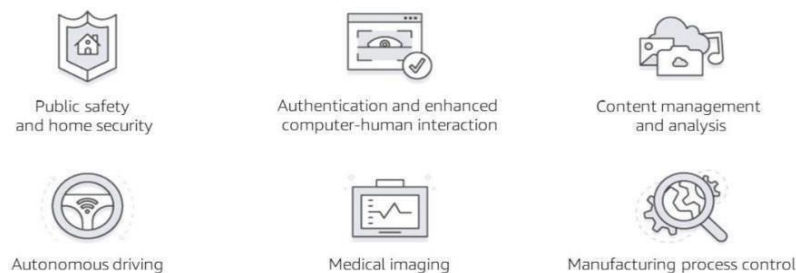


Fig 2.3 Applications of Computer Vision:

### **Public safety and home security**

Computer vision with image and facial recognition can help to quickly identify unlawful entries or persons of interest. This process can result in safer communities and a more effective way of deterring crimes.

### **Authentication and enhanced computer-human interaction**

Enhanced human-computer interaction can improve customer satisfaction. Examples include products that are based on customer sentiment analysis in retail outlets or faster banking services with quick authentication that is based on customer identity and preferences.

### **Content management and analysis**

Millions of images are added every day to media and social channels. Using computer vision technologies such as metadata extraction and image classification can improve efficiency and revenue opportunities. By using computer-vision technologies, auto manufacturers can provide improved and safer self-driving car navigation, which can help realize autonomous driving and make it a reliable transportation option.

## Medical imaging

Medical image analysis with computer vision can improve the accuracy and speed of a patient's medical diagnosis, which can result in better treatment outcomes and life expectancy. Manufacturing process control Well-trained computer vision that is incorporated into robotics can improve quality assurance and operational efficiencies in manufacturing applications.

### Computer vision problems:

**Problem 01:** Recognizing food & state whether it's breakfast or lunch or dinner



Fig 2.4: Recognizing food & state

The CV classified the objects as milk, peaches, ice cream, salad and nuggets, thus, it's a breakfast

## Problem 02: Video Analysis



Fig 2.5 Content recognition in Video Analysis

### 1. Image and Video Analysis

Amazon Recognition is a computer vision service based on deep learning. You can use it to add image and video analysis to your applications.

Amazon Recognition enables you to perform the following types of analysis: **Searchable image and video libraries**—Amazon Recognition makes images and stored videos searchable so that you can discover the objects and scenes that appear in them.

**Face-based user verification**—Amazon Recognition enables your applications to confirm user identities by comparing their live image with a reference image. **Sentiment and demographic analysis**—Amazon Recognition interprets emotional expressions, such as happy, sad, or surprise. It can also interpret demographic information from facial images, such as gender.

**Unsafe content detection**—Amazon Recognition can detect inappropriate content in images and in stored videos.

**Text detection**—Amazon Recognition Text in Image enables you to recognize and extract text content from the image

## CHAPTER 3 REFLECTION ON LEARNING

### Get started with product image search

Add ML Kit Object Detection and Tracking API to the project

#### Add the dependencies for ML Kit Object Detection and Tracking

The ML Kit dependencies allow you to integrate the ML Kit ODT SDK in your app.

Go to the `app/build.gradle` file of your project and confirm that the dependency is already there:

```
dependencies {
```

```
// ...
```

```
implementation 'com.google.mlkit:object-detection:16.2.4'
```

```
}
```

### 1. Sync your project with gradle files

To be sure that all dependencies are available to your app, you should sync your project with gradle files at this point.

Select **Sync Project with Gradle Files** (  ) from the Android Studio toolbar.

(If this button is disabled, make sure you import only `starter/app/build.gradle`, not the entire)

### 2. Run the starter app

Now that you have imported the project into Android Studio and added the dependencies for ML Kit Object Detection and Tracking, you are ready to run the app for the first time.

Connect your Android device via USB to your host or [Start the Android Studio emulator](#), and click

**Run** (  ) in the Android Studio toolbar.

### 3. Run and explore the app

The app should launch on your Android device. It has some boilerplate code to allow you to capture a photo, or select a preset image, and feed it to an object detection and tracking pipeline that you'll build in this codelab. Explore the app a little bit before writing code:

First, there is a **Button** ( ) at the bottom to

- launch the camera app integrated in your device/emulator

- take a photo inside your **camera app**
- **receive** the captured image in **starter app**
- display the image

Try out the "**Take photo**" button. Follow the prompts to take a photo, **accept** the photo and observe it displayed inside the **starter app**.

Second, there are 3 preset images that you can choose from. You can use these images later to test the object detection code if you are running on an Android emulator.

1. **Select** an image from the 3 preset images.
2. See that the image shows up in the larger view.



Fig-3.1 Product Image

### Search Add on-device object detection

In this step, you'll add the functionality to the starter app to detect objects in images. As you saw in the previous step, the starter app contains boilerplate code to take photos with the camera app on the device. There are also 3 preset images in the app that you can try object detection on, if you are running the codelab on an Android emulator.

When you select an image, either from the preset images or by taking a photo with the camera app, the boilerplate code decodes that image into a **Bitmap** instance, shows it on the screen and

calls the `runObjectDetection` method with the image. In this step, you will add code to the `runObjectDetection` method to do object detection!

Set up and run on-device object detection on an image

There are only 3 simple steps with 3 APIs to set up ML Kit ODT

- prepare an image: `InputImage`
- create a detector object: `ObjectDetection.getClient(options)`
- connect the 2 objects above: `process(image)`

You achieve these inside the function `**runObjectDetection(bitmap: Bitmap)**` in file `MainActivity.kt`.

```
/**  
 * ML Kit Object Detection Function  
 */ private fun runObjectDetection(bitmap: Bitmap)  
{  
}
```

Right now the function is empty. *Move on to the following steps to integrate ML Kit ODT!* Along the way, Android Studio would prompt you to add the necessary imports

- `com.google.mlkit.vision.common.InputImage`
- `com.google.mlkit.vision.objects.ObjectDetection`
- `com.google.mlkit.vision.objects.defaults.ObjectDetectorOptions`

### Step 1: Create an `InputImage`

ML Kit provides a simple API to create an `InputImage` from a `Bitmap`. Then you can feed an `InputImage` into the ML Kit APIs.

```
// Step 1: create ML Kit's InputImage object val  
image = InputImage.fromBitmap(bitmap, 0)
```

**Add** the above code to the top of `runObjectDetection(bitmap:Bitmap)`.

Step 2: Create a detector instance

ML Kit follows *Builder Design Pattern*, you would pass the configuration to the builder, then acquire a **detector** from it. There are 3 options to configure (the one in **bold** is used in codelab):

- detector mode (***single image** or stream*)
- detection mode (*single or **multiple** object detection*)
- classification mode (***on** or off*)

This codelab is for single image - multiple object detection & classification, let's do that:

```
// Step 2: acquire detector object val options =  
ObjectDetectorOptions.Builder()  
    .setDetectorMode(ObjectDetectorOptions.SINGLE_IMAGE_MODE)  
    .enableMultipleObjects()  
    .enableClassification()  
    .build()          val          objectDetector  
= ObjectDetection.getClient(options)
```

Step 3: Feed image(s) to the detector Object detection and classification is async processing:

- you send an image to detector (via `process()`)
- detector works pretty hard on it
- detector reports the result back to you via a callback

The following code does just that (**copy and append** it to the existing code inside `fun runObjectDetection(bitmap:Bitmap)`):

```
// Step 3: feed given image to detector and setup callback objectDetector.process(image)  
    .addOnSuccessListener {  
        // Task completed successfully debugPrint(it)  
    }  
    .addOnFailureListener {  
        // Task failed with an exception  
        Log.e(TAG, it.message.toString())  
    }
```

Upon completion, detector notifies you with


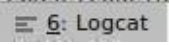
1. Total number of objects detected
2. Each detected object is described with
  - `trackingId`: an integer you use to track it cross frames (NOT used in this codelab)
  - `boundingBox`: object's bounding box
  - `labels`: list of label(s) for the detected object (only when classification is enabled)
  - `index` (Get the index of this label)
  - `text` (Get the text of this label including "Fashion Goods", "Food", "Home Goods", "Place", "Plant")
  - `confidence` (a float between 0.0 to 1.0 with 1.0 means 100%)

You have probably noticed that the code prints detected results to Logcat with `debugPrint()`. Add it into `MainActivity` class:

```
private fun debugPrint(detectedObjects: List<DetectedObject>) {  
  
    detectedObjects.forEachIndexed { index, detectedObject -> val  
        box = detectedObject.boundingBox  
  
        Log.d(TAG, "Detected object: $index")  
        Log.d(TAG, " trackingId: ${detectedObject.trackingId}")  
        Log.d(TAG, " boundingBox: (${box.left}, ${box.top}) - (${box.right},${box.bottom})")  
        detectedObject.labels.forEach {  
            Log.d(TAG, " categories: ${it.text}")  
            Log.d(TAG, " confidence: ${it.confidence}") }  
        }  
    }  
}
```



Now you are ready to accept images for detection!

Run the codelab by clicking **Run** (  ) in the Android Studio toolbar. Try selecting a preset image or take a photo, then look at the *log* IDE  .Now open this.

D/MLKit Object Detection: Detected object:

0 D/MLKit Object Detection: trackingId:

null

D/MLKit Object Detection: boundingBox: (481, 2021) - (2426,3376)

D/MLKit Object Detection: categories: Fashion good

D/MLKit Object Detection: confidence:

0.90234375 D/MLKit Object Detection: Detected

object: 1 D/MLKit Object Detection: trackingId:

null

D/MLKit Object Detection: boundingBox: (2639, 2633) - (3058,3577)

D/MLKit Object Detection: Detected object: 2

D/MLKit Object Detection: trackingId: null

D/MLKit Object Detection: boundingBox: (3, 1816) - (615,2597)

D/MLKit Object Detection: categories: Home good

D/MLKit Object Detection: confidence: 0.75390625

which means that detector saw 3 objects of:

- categories are **Fashion good** and **Home good**.
- there is no category returned for the 2nd because it is an unknown class.
- no `trackingId` (because this is single image detection mode)
- position inside the `boundingBox` rectangle (e.g. (481, 2021) – (2426, 3376))
- detector is pretty confident that the 1st is a **Fashion good** (90%) (*it is a dress*)

**Technically** that is all that you need to get ML Kit Object Detection to work— you got it all at this moment! **Congratulations!**

Yeah, on the UI side, you are still at the stage when you started, but you could make use of the detected results on the UI such as drawing out the bounding box to create a better experience. The next step is to visualize the detected results!

## Post-processing the detection results

In previous steps, you printed the detected result into *logcat*: simple and fast.

In this section, you will make use of the result in the image:

- draw the bounding box on image
- draw the category name and confidence inside bounding box

### **Understand the visualization utilities**

There is some boilerplate code inside the codelab to help you visualize the detection result.

Leverage these utilities to make our visualization code simple:


- `class ImageClickableView` This is an image view class that provides some convenient utils for visualization and interaction with the detection result
- `fun drawDetectionResults(results: List<DetectedObject>)` This method draws white circles at the center of each object detected.
- `fun setOnObjectClickListener(listener: ((objectImage: Bitmap) -> Unit))` This is a callback to receive the cropped image that contains only the object that the user has tapped on. You will send this cropped image to the image search backend in a later codelab to get a visually similar result. In this codelab, you won't use this method yet.

### **Show the ML Kit detection result**

Use the visualization utilities to show the ML Kit object detection result on top of the input image.

Go to where you call `debugPrint()` and add the following code snippet below it:

```
runOnUiThread { viewBinding.ivPreview.drawDetectionResults(it)
}
```

Now click **Run** (  ) in the Android Studio toolbar

Once the app loads, press the Button with the camera icon, point your camera to an object, take a photo, accept the photo (in Camera App) or you can easily tap any preset images. You should see

the detection result; press the Button again or select another image to repeat a couple of times, and experience the latest ML Kit ODT!

### **Further with product image search**

[Build a product image search backend with Vision API Product Search](#)

### **About Vision API Product Search**

[Vision API Product Search](#) is a feature in Google Cloud that allows retailers to create products, each containing reference images that visually describe the product from a set of viewpoints. Retailers can then add these products to product sets. Currently Vision API Product Search supports the following product categories: homegoods, apparel, toys, packaged goods, and general.

When users query the product set with their own images, Vision API Product Search applies machine learning to compare the product in the user's query image with the images in the retailer's product set, and then returns a ranked list of visually and semantically similar results.

### Setup API key

In the Vision API Product Search quickstart, you have built a product search backend that can take a query image and return visually similar products. In order to call the product search API from a mobile app, you will need to setup an API key and then restrict access of the API key to your own mobile apps, in order to avoid unauthorized use.

#### Create an API key

1. Go to **Cloud Console > APIs & Services > Credentials**. You can also click on this [URL](#) and select the project that you have used in the Product Search quickstart.
2. Select **Create Credentials > API key**. You will see this dialog if your API key has been created successfully:

Take note of this API key. You will use it later in this codelab.

#### **Restrict access to the API key**

When seeing the prompt above, select **Restrict key**.

Follow the instructions on the screen to apply these restrictions:

- Application restrictions > Android apps
- API restrictions > Restrict key > Cloud Vision API

Restrict access to the API key is **strongly recommended** to protect your API key from unauthorized access in production.

### 6. Update the API endpoints

## Change the API configurations

Go to the `ProductSearchAPIClient` class and you will see the configs of the product search backend already defined. Comment out the configs of the demo backend:

```
// Define the product search backend
// Option 1: Use the demo project that we have already deployed for you
// const val VISION_API_URL =
    "https://us-central1-odml-codelabs.cloudfunctions.net/productSearch"
// const val VISION_API_KEY = ""
// const val VISION_API_PROJECT_ID = "odml-codelabs"
// const val VISION_API_LOCATION_ID = "us-east1"
// const val VISION_API_PRODUCT_SET_ID = "product_set0"
```

Then replace them with your config:


// Option 2: Go through the Vision API Product Search quickstart and deploy to your project.

// Fill in the const below with your project info.

```
const val VISION_API_URL = "https://vision.googleapis.com/v1"
const val VISION_API_KEY = "YOUR_API_KEY" const val
VISION_API_PROJECT_ID = "YOUR_PROJECT_ID"

const val VISION_API_LOCATION_ID = "YOUR_LOCATION_ID"const val
VISION_API_PRODUCT_SET_ID = "YOUR_PRODUCT_SET_ID"
```

- **VISION\_API\_URL** is the API endpoint of Cloud Vision API.
- **VISION\_API\_KEY** is the API key that you created earlier in this codelab.
- **VISION\_API\_PROJECT\_ID , VISION\_API\_LOCATION\_ID , VISION\_API\_PRODUCT\_SET\_ID** is the value you used in the Vision API Product Search quickstart earlier in this codelab.

Now click **Run** (  in the Android Studio toolbar. Once the app loads, tap any preset images, select an detected object, tap the **Search** button to see the search results. The app is now using the product search backend that you have just created!

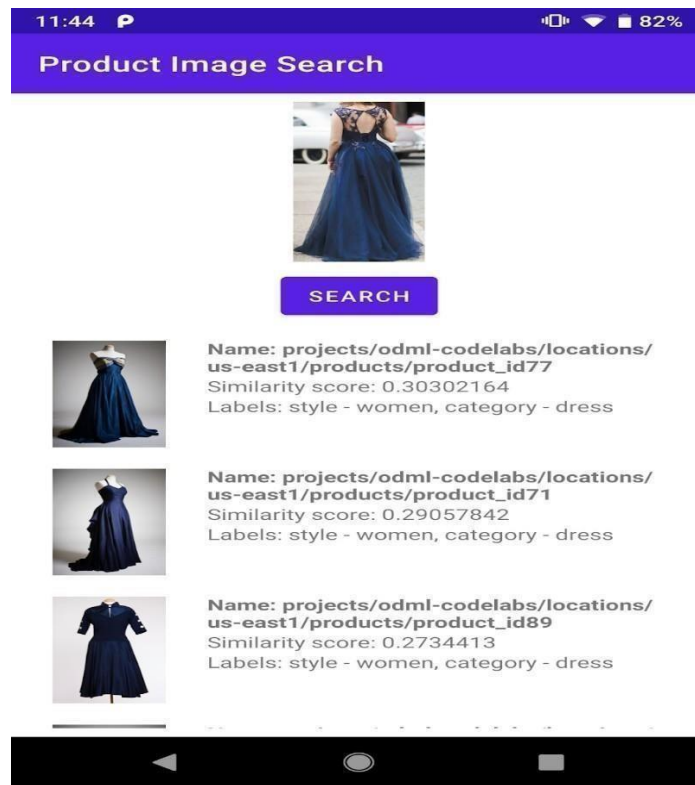


Fig-3.2 Further Classification of Product Image Search

## Object detection

### Add ML Kit Object Detection and Tracking API to the project

### Add the dependencies for ML Kit Object Detection and Tracking

The ML Kit dependencies allow you to integrate the ML Kit ODT SDK in your app. Add the following lines to the end of the `app/build.gradle` file of your project:

#### **build.gradle**

```
dependencies {

    // ...

    implementation 'com.google.mlkit:object-detection:16.2.4'

}
```

#### **Sync your project with gradle files**

To be sure that all dependencies are available to your app, you should sync your project with

gradle files at this point. Select **Sync Project with Gradle Files** (  ) from the Android Studio toolbar

(If this button is disabled, make sure you import only `starter/app/build.gradle` , not the entire repository.)

#### 4. Run the starter app


Now that you have imported the project into Android Studio and added the dependencies for ML Kit Object Detection and Tracking, you are ready to run the app for the first time.

Connect your Android device via USB to your host, or Start the Android Studio emulator, and click

**Run** (  ) in the Android Studio toolbar.

#### **Run and explore the app**

The app should launch on your Android device. It has some boilerplate code to allow you to capture a photo, or select a preset image, and feed it to an object detection and tracking pipeline that you'll build in this codelab. Let's explore the app a little bit before writing code.

First, there is a **Button** (  ) at the bottom to:

- bring up the camera app integrated in your device/emulator
- take a photo inside your **camera app**
- **receive** the captured image in **starter app**
- display the image

Try out the **Take photo** button, follow the prompts to take a photo, **accept** the photo and observe it displayed inside the *starter app*.

Repeat a few times to see how it works:

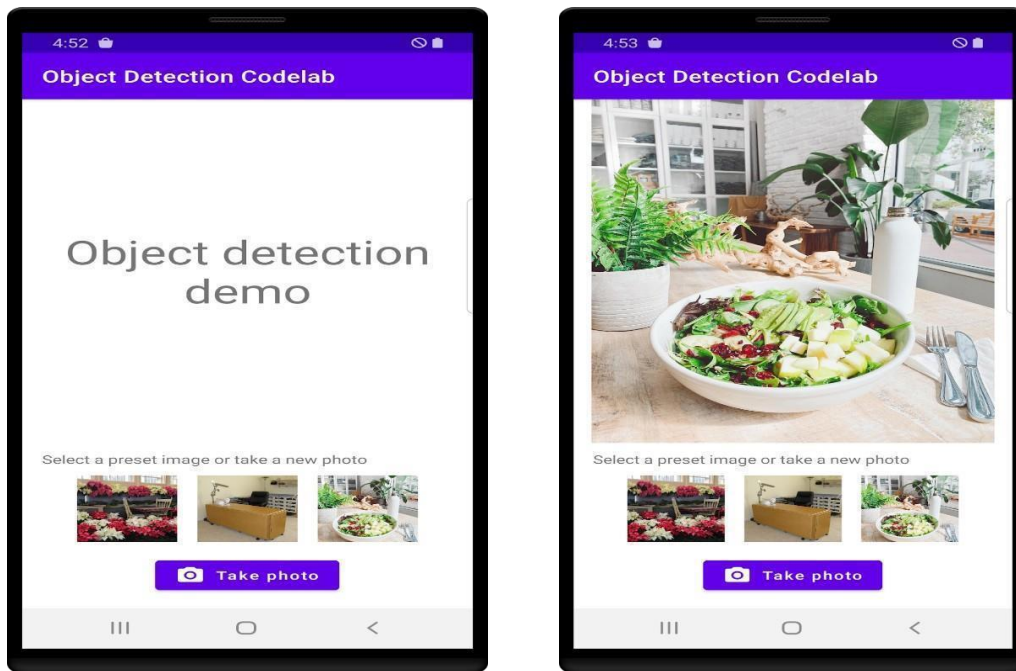


Fig- 3.3 Sample images for the object detection

Second, there are 3 preset images that you can choose from. You can use these images later to test the object detection code if you are running on an Android emulator.

Select an image from the 3 preset images. See that the image shows up in the larger view:

### Post-processing the detection results

In previous steps, you print the detected result into *logcat*: simple and fast.

In this section, you'll make use of the result into the image:

- draw the bounding box on image
- draw the category name and confidence inside bounding box

### **Understand the visualization utilities**

There is some boilerplate code inside the codelab to help you visualize the detection result.

Leverage these utilities to make our visualization code simple:



- `data class BoxWithText(val box: Rect, val text: String)` This is a data class to store an object detection result for visualization. `box` is the bounding box where the object locates, and `text` is the detection result string to display together with the object's bounding box.
- `fun drawDetectionResult(bitmap: Bitmap, detectionResults: List<BoxWithText>): Bitmap` This method draws the object detection results in `detectionResults` on the input `bitmap` and returns the modified copy of it.

Here is an example of an output of the `drawDetectionResult` utility method:

## Visualize the ML Kit detection result

Use the visualization utilities to draw the ML Kit object detection result on top of the input image.

Go to where you call `debugPrint()` and add the following code snippet below it:

```
// Parse ML Kit's DetectedObject and create corresponding visualization
data val detectedObjects = it.map { obj -> var text = "Unknown"

    // We will show the top confident detection result if it exist if
    (obj.labels.isNotEmpty()) { val firstLabel = obj.labels.first() text =
        "${firstLabel.text}, ${firstLabel.confidence.times(100).toInt()}%"
    }

    BoxWithText(obj.boundingBox, text)
val visualizedResult = drawDetectionResult(bitmap, detectedObjects)

// Show the detection result on the app screen
runOnUiThread
{ imageView.setImageBitmap(visualizedResult)
}
```

- You start by parsing the ML Kit's `DetectedObject` and creating a list of `BoxWithText` objects to display the visualization result.
- Then you draw the detection result on top of the input image, using the

`drawDetectionResult` utility method, and show it on the screen.

**Run it:** Now click **Run** (  ) in the Android Studio toolbar.

Once the app loads, press the Button with the camera icon, point your camera to an object, take a photo, accept the photo (in Camera App) or you can easily tap any preset images. You should see the detection results; press the Button again or select another image to repeat a couple of times to experience the latest ML Kit ODT!



Fig- 3.4 Detection of objects for input image

## **Further with object detection**

Build and deploy a custom object-detection model with TensorFlow Lite

### Object Detection

Object detection is a set of computer vision tasks that can detect and locate objects in a digital image. Given an image or a video stream, an object detection model can identify which of a known set of objects might be present, and provide information about their positions within the image.

## TensorFlow Lite

TensorFlow Lite is a cross-platform machine learning library that is optimized for running machine learning models on edge devices, including Android and iOS mobile devices.

TensorFlow Lite is actually the core engine used inside ML Kit to run machine learning models. There are two components in the TensorFlow Lite ecosystem that make it easy to train and deploy machine learning models on mobile devices:

- *Model Maker* is a Python library that makes it easy to train TensorFlow Lite models using your own data with just a few lines of code, no machine learning expertise required.
- *Task Library* is a cross-platform library that makes it easy to deploy TensorFlow Lite models with just a few lines of code in your mobile apps.

This codelab focuses on TFLite. Concepts and code blocks that are not relevant to TFLite and object detection are not explained and are provided for you to simply copy and paste.

### Understand the starter app

In order to keep this codelab simple and focused on the machine learning bits, the starter app contains some boilerplate code that do a few things for you:

- It can take photos using the device's camera.
- It contains some stock images for you to try out object detection on an Android emulator.
- It has a convenient method to draw the object detection result on the input bitmap.
- `fun runObjectDetection(bitmap: Bitmap)` This method is called when you choose a preset image or take a photo. `bitmap` is the input image for object detection. Later in this codelab, you will add object detection code to this method.
- `data class DetectionResult(val boundingBoxes: Rect, val text: String)` This is a data class that represents an object detection result for visualization. `boundingBoxes` is the rectangle where the object locates, and `text` is the detection result string to display together with the object's bounding box.
- `fun drawDetectionResult(bitmap: Bitmap, detectionResults: List<DetectionResult>):`

**Bitmap** This method draws the object detection results in **detectionResults** on the input **bitmap** and returns the modified copy of it.

Here is an example of an output of the **drawDetectionResult** utility method.

```
detectionResults =  
[ { boundingBoxes: ((200,220), (1289, 1264)),  
  text: "Cup (82%) " } ]
```

bitmap =

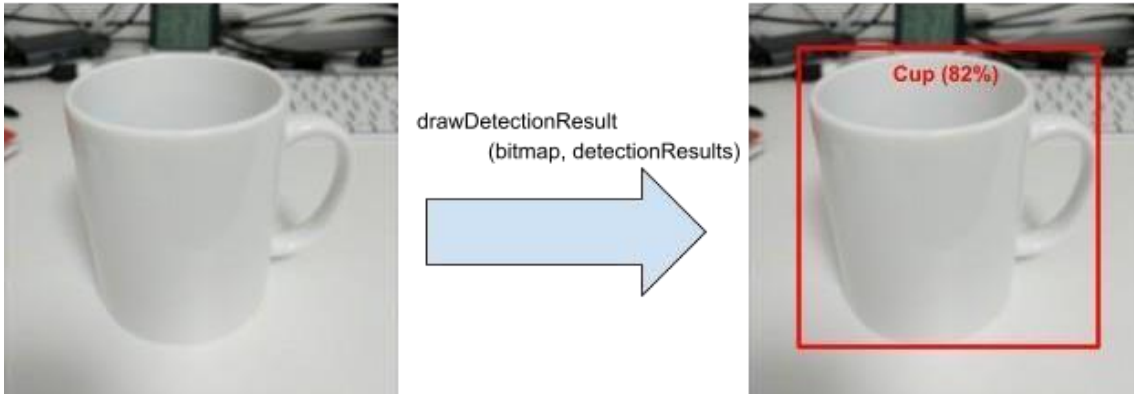


Figure3.5:Bit map for the object.

- **Add on-device object detection**

Now you'll build a prototype by integrating a pre-trained TFLite model that can detect common objects into the starter app.

### **Download a pre-trained TFLite object detection model**

There are several object detector models on TensorFlow Hub that you can use. For this codelab, you'll download the EfficientDet-Lite Object detection model, trained on the COCO 2017 dataset, optimized for TFLite, and designed for performance on mobile CPU, GPU, and EdgeTPU.

file downloadDownload the model

Next, use the TFLite Task Library to integrate the pre-trained TFLite model into your starter app. The TFLite Task Library makes it easy to integrate mobile-optimized machine learning models into a mobile app. It supports many popular machine learning use cases, including object detection, image classification, and text classification. You can load the TFLite model and run it with just a few lines of code.

TFLite Task Library only supports TFLite models that contain valid metadata. You can find more supported object detection models from this TensorFlow Hub collection.

## Add the model to the starter app


1. Copy the model that you have just downloaded to the `assets` folder of the starter app. You can find the folder in the **Project** navigation panel in Android Studio.
2. Name the file `model.tflite`.

## Update the Gradle file Task Library dependencies

Go to the `app/build.gradle` file and add this line into the `dependencies` configuration:

```
implementation 'org.tensorflow:tensorflow-lite-task-vision:0.3.1'
```

## Sync your project with gradle files:

To be sure that all dependencies are available to your app, you should sync your project with gradle files at this point. Select **Sync Project with Gradle Files** (  ) from the Android Studio toolbar.

**(If this button is disabled, make sure you import only starter/app/build.gradle, not the entire repository.)**

## Set up and run on-device object detection on an image

There are only 3 simple steps with 3 APIs to load and run an object detection model:

- prepare an image / a stream: `TensorImage`
- create a detector object: `ObjectDetector`
- connect the 2 objects above: `detect(image)`

You achieve these inside the function `runObjectDetection(bitmap: Bitmap)` in file `MainActivity.kt`.

```
/**
 * TFLite Object Detection Function
 */
private fun runObjectDetection(bitmap: Bitmap)
{ //TODO: Add object detection code here
}
```

Right now the function is empty. Move on to the following steps to implement the TFLite object

detector. Along the way, Android Studio will prompt you to add the necessary imports:

- `org.tensorflow.lite.support.image.TensorImage`
- `org.tensorflow.lite.task.vision.detector.ObjectDetector`

**Create Image Object** The images you'll use for this codelab are going to come from either the on- device camera, or preset images that you select on the app's UI. The input image is decoded into the `Bitmap` format and passed to the `runObjectDetection` method.

TFLite provides a simple API to create a `TensorImage` from `Bitmap`. Add the code below to the top of `runObjectDetection(bitmap:Bitmap)`:

```
// Step 1: create TFLite's TensorImage object val  
image = TensorImage.fromBitmap(bitmap)
```

## Create a Detector instance

TFLite Task Library follows the Builder Design Pattern. You pass the configuration to a builder, then acquire a detector from it. There are several options to configure, including those to adjust the sensitivity of the object detector:

- max result (the maximum number of objects that the model should detect)
- score threshold (how confidence the object detector should be to return a detected object)
- label allowlist/denylist (allow/deny the objects in a predefined list)

Initialize the object detector instance by specifying the TFLite model file name and the configuration options:

```
// Step 2: Initialize the detector object val options =  
ObjectDetector.ObjectDetectorOptions.builder()  
    .setMaxResults(5)  
    .setScoreThreshold(0.5f)  
    .build()          val          detector          =  
ObjectDetector.createFromFileAndOptions( this, // the  
application context  
    "model.tflite", // must be same as the filename in assets folder options)
```

## Feed Image(s) to the detector

Add the following code to `fun run ObjectDetection(bitmap:Bitmap)`. This will feed your images to the detector.

```
// Step 3: feed given image to the model and print the detection result val
results = detector.detect(image)
```

Upon completion, the detector returns a list of `Detection`, each containing information about an object that the model has found in the image. Each object is described with:

- `boundingBox`: the rectangle declaring the presence of an object and its location within the image
- `categories`: what kind of object it is and how confident the model is with the detection result. The model returns multiple categories, and the most confident one is first.
- `label`: the name of the object category.
- `classificationConfidence`: a float between 0.0 to 1.0, with 1.0 representing 100%

## Print the detection results

Add the following code to `fun runObjectDetection(bitmap:Bitmap)`. This calls a method to print the object detection results to Logcat.

```
// Step 4: Parse the detection result and show it debugPrint(results)
```

Then add this `debugPrint()` method to the `MainActivity` class

```
private fun debugPrint(results : List<Detection>)
```


```
{ for ((i, obj) in results.withIndex()) { val box =
obj.boundingBox
    Log.d(TAG, "Detected object: ${i} ")
    Log.d(TAG, " boundingBox: (${box.left}, ${box.top}) - (${box.right},${box.bottom})")

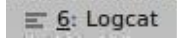
    for ((j, category) in obj.categories.withIndex())
        { Log.d(TAG, " Label $j: ${category.label}") val
```

```

        confidence: Int = category.score.times(100).toInt()
        Log.d(TAG, " Confidence: ${confidence}%")
    }
}
}

```

Now your object detector is ready! Compile and run the app by clicking **Run**  ( ) in the Android Studio toolbar. Once the app has shown up on the device, tap on any of the preset images

to start the object detector. Then look at the **Logcat** window\*( **Logcat** \*)\* inside your IDE, and you should see something similar to this:

```

D/TFLite-ODT: Detected object: 0
D/TFLite-ODT: boundingBox: (0.0, 15.0) - (2223.0,1645.0)
D/TFLite-ODT:   Label 0: dining
table D/TFLite-ODT:   Confidence:
77% D/TFLite-ODT: Detected object:
1
D/TFLite-ODT: boundingBox: (702.0, 3.0) - (1234.0,797.0)
D/TFLite-ODT:   Label 0: cup
D/TFLite-ODT:   Confidence:
69%

```

The model is confident that the 1st is a **dining table** (77%)

However, on the UI side, you are still at the starting point. Now you have to make use of the detected results on the UI, by post-processing the detected results.

Draw the detection result on the input image

In previous steps, you printed the detection result into **logcat**: simple and fast. In this step, you'll make use of the utility method already implemented for you in the starter app, in order to:



- draw a bounding box on an image
- draw a category name and confidence percentage inside the bounding box


1. Replace the `debugPrint(results)` call with the following code snippet:

```
val resultToDisplay = results.map {
    // Get the top-1 category and craft the display text val
    category = it.categories.first()
    val text = "${category.label}, ${category.score.times(100).toInt()}%"

    // Create a data object to display the detection result DetectionResult(it.boundingBox,
    text)
}
// Draw the detection result on the bitmap and show it. val
imgWithResult = drawDetectionResult(bitmap, resultToDisplay)
runOnUiThread {
    imageView.setImageBitmap(imgWithResult)
}
```

**Note:** Because object detection is a computation intensive process and TFLite Task Library runs on the same thread where it is called, it's important to run it on a background thread to avoid blocking the app UI. You can see that we use Kotlin coroutine to call the `runObjectDetection` method

```
lifecycleScope.launch(Dispatchers.Default) { runObjectDetection(bitmap) }
```

1. Now click **Run** (  ) in the Android Studio toolbar. 3. Once the app loads, tap on one of the preset images to see the detection result.

Want to try with your own photo? Tap on the **Take photo** button and capture some pictures of objects around you.

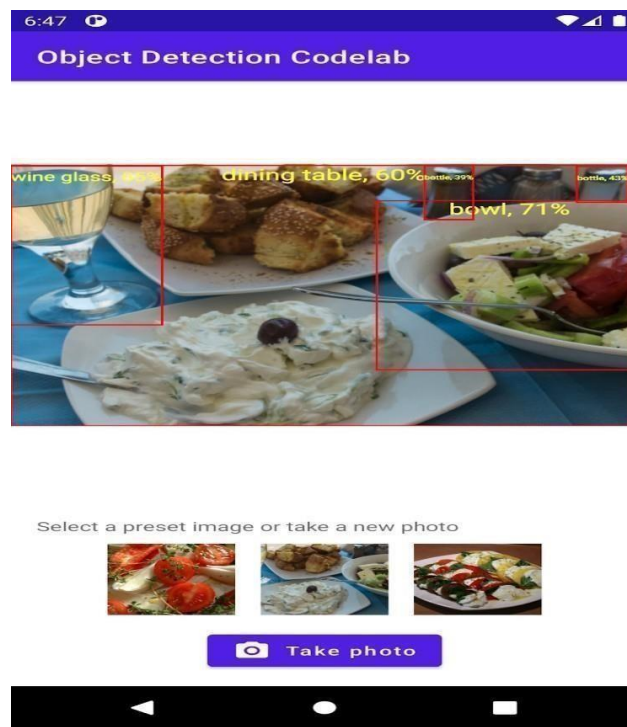


Fig- 3.5 further detection of object for the input image

## Image Classification

First, you'll need the app from the Build your first Computer Vision App on Android or iOS Codelab. If you have gone through the lab, it will be called ImageClassifierStep1. If you don't want to go through the lab, you can clone the finished version from the [repo](#). Open it in Android Studio, do whatever updates you need, and when it's ready run the app to be sure it works. You should see something like this:



Fig -3.6 Sample for the image classification

It's quite a primitive app, but it shows some very powerful functionality with just a little code. However, if you want this flower to be recognized as a daisy, and not just as a flower, you'll have to update the app to use your custom model from the Create a custom model for your image classifier codelab.

## Create a custom model for your image classification

All of the code to follow along has been prepared for you and is available to execute using Google Colab. If you don't have access to Google Colab, you can clone the repo and use the notebook called `CustomImageClassifierModel.ipynb` which can be found in the `ImageClassificationMobile->colab` directory.

### 1. Install and import dependencies

Install TensorFlow Lite Model Maker. You can do this with a pip install. The `&> /dev/null` at the end just suppresses the output. Model Maker outputs a lot of stuff that isn't immediately relevant. It's been suppressed so you can focus on the task at hand.

```
# Install Model maker
```

```
!pip install -q tflite-model-maker &> /dev/null
```

Next you'll need to import the libraries that you need to use and ensure that you are using TensorFlow 2.x:

```
# Imports and check that we are using TF2.x
```

```
import numpy as np import os
```

```
from tflite_model_maker import configs from
```

```
tflite_model_maker import ExportFormat from
```

```
tflite_model_maker import model_spec from
```

```
tflite_model_maker import image_classifier
```

```
from tflite_model_maker.image_classifier import DataLoader
```

```
import tensorflow as tf assert tf.
```

```
version_____
```

```
.startswith('2')
```

```
tf.get_logger().setLevel('ERROR'
```

```
)
```

Now that the environment is ready, it's time to start creating your model!

## **2. Download and Prepare your Data**

If your images are organized into folders, and those folders are zipped up, then if you download the zip and decompress it, you'll automatically get your images labelled based on the folder they're in. This directory will be referenced as `data_path`.

```
data_path = tf.keras.utils.get_file('flower_pics', untar=True)
```

```
'https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz',
```

This data path can then be loaded into a neural network model for training with TensorFlow Lite Model Maker's `ImageClassifierDataLoader` class. Just point it at the folder and you're good to go. One important element in training models with machine learning is to not use *all* of your data for training. Hold back a little to test the model with data it hasn't previously seen. This is easy to do with the `split` method of the dataset that comes back from `ImageClassifierDataLoader`. By passing a 0.9 into it, you'll get 90% of it as your training data, and 10% as your test data:

```
data = DataLoader.from_folder(data_path)
```

```
train_data, test_data = data.split(0.9)
```

Now that your data is prepared, you can create a model using it.

## **3. Create the Image Classifier Model**

Model Maker abstracts a lot of the specifics of designing the neural network so you don't have to deal with network design, and things like convolutions, dense, relu, flatten, loss functions and optimizers. For a default model, you can simply use a single line of code to create a model by training a neural network with the provided data:

```
model = image_classifier.create(train_data)
```

When you run this, you'll see output that looks a bit like the following: Model:

"sequential\_2"

---

Layer (type)	Output Shape	Param #
=====		
hub_keras_layer_v1v2_2 (HubK	(None, 1280)	3413024
<hr/>		
dropout_2 (Dropout)	(None, 1280)	0
<hr/>		
dense_2 (Dense)	(None, 5)	6405
=====		
Total params: 3,419,429		
Trainable params: 6,405		
Non-trainable params: 3,413,024		
<hr/>		

Epoch 1/5

103/103 [===] - 15s 129ms/step - loss: 1.1169 - accuracy: 0.6181

Epoch 2/5

103/103 [===] - 13s 126ms/step - loss: 0.6595 - accuracy: 0.8911

Epoch 3/5

103/103 [===] - 13s 127ms/step - loss: 0.6239 - accuracy: 0.9133

Epoch 4/5

103/103 [===] - 13s 128ms/step - loss: 0.5994 - accuracy:

0.9287 Epoch 5/5

103/103 [===] - 13s 126ms/step - loss: 0.5836 - accuracy: 0.9385

The first part is showing your model architecture. What Model Maker is doing behind the scenes is called *Transfer Learning*, which uses an existing pre-trained model as a starting point, and just taking the things that that model learned about how images are constructed and applying them to understanding these 5 flowers.

The key is the word 'Hub', telling us that this model came from TensorFlow Hub. By default, TensorFlow Lite Model Maker uses a model called 'MobileNet' which is designed to recognize 1000 types of image. The logic here is that the methodology that it uses, by learning 'features' to distinguish between 1000 classes, can be reused. The same 'features' can be mapped to our 5 classes of flowers, so they don't have to be learned from scratch.

The model went through 5 epochs – where an epoch is a full cycle of training where the neural network tries to match the images to their labels. By the time it went through 5 epochs, in around 1 minute, it was 93.85% accurate on the training data. Given that there's 5 classes, a random guess would be 20% accurate, so that's progress! (It also reports a 'loss' number, but you safely ignore that for now.)

Earlier you split the data into training and test data, so you can get a gauge for how the network performs on data it hasn't previously seen – a better indicator of how it might perform in the real world by using `model.evaluate` on the test data:

```
loss, accuracy = model.evaluate(test_data)
```

This will output something like this:

```
12/12 [===] - 5s 115ms/step - loss: 0.6622 - accuracy: 0.8801
```

Note the accuracy here. It's 88.01%, so using the default model in the real world should expect that level of accuracy. That's not bad for the default model that you trained in about a minute.

Of

course you could probably do a lot of tweaking to improve the model, and that's a science unto itself!

#### **4. Export the Model**

Now that the model is trained, the next step is to export it in the .tflite format that a mobile application can use. Model maker provides an easy export method that you can use — simply specify the directory to output to.

Here's the code: `model.export(export_dir='/mm_flowers')`



```
[9] Epoch 1/5
    103/103 [=====] - 15s
    Epoch 2/5
    103/103 [=====] - 13s
    Epoch 3/5
    103/103 [=====] - 13s
    Epoch 4/5
    103/103 [=====] - 13s
    Epoch 5/5
    103/103 [=====] - 13s

[10] loss, accuracy = model.evaluate(test_data)
```

Fig – 3.7 Performance metrics for the image classification

## CONCLUSION

These chapters described how model explain ability relates to AI/ML solutions, giving customers insight to explain ability requirements when initiating AI/ML use cases. Four pillars were presented to assess model explanation ability options to bridge knowledge gaps and requirements for simple to complex algorithms. To help convey how these models explain ability options relate to real-world scenarios, examples from a range of industries were demonstrated. It is recommended that AI/ML owners or business leaders follow these steps when initiating a new AI/ML solution:

- Collect business requirements to identify the level of explainability required for your business to accept the solution.
- Based on business requirements, implement an assessment for model explanation ability.
- Work with an AI/ML technician to communicate the model, explain ability assessment and find the optimal AI/ML solution to meet your business objectives.
- After the solution is completed, revisit the model and explain ability assessment to evaluate that business requirements are continuously met.
- By taking these steps, we will mitigate regulation risks and ensure trust in our model. With this trust, when the event comes to push your AI/ML solution into a Google production environment, we will be ready to create business value for our use case.



