Getting Started with ML Kit

SierraOBryan

SIERRA OBRYAN (SHE/HER)

@_sierraOBryan (she/her)

Prerequisites Equipment:

- Bring your laptop; must be able to connect to local conference Wi-Fi.
- Any web browser; during the workshop, we will be using a browser based model maker

Installations:

- Latest and Greatest Android Studio: https://developer.android.com/studio

Once you have Android Studio downloaded, please open the editor and go through the setup wizard and create an emulator. Recommendation: Pixel 6 Emulator with API 33 (Android 13) installed but any phone with API level 24+ will do; see below for a help guide to set this up!

Helpful codelabs to guide you through this process:

- Download and Install Android Studio
- Running the Android Emulator
- [Optional] Connecting your own device

No knowledge of Kotlin, Android, Android Studio is required or expected (although welcome!). Using an Android Phone is also not required! Everyone is welcome (:

Agenda

Introduction to ML Kit, Android, and Compose,

Building UI with Jetpack Compose

Implementing the ML Kit

Training and using a custom model

Adding CameraX Support

Choose your own Adventure

What is it??

“ML Kit brings Google’s machine learning expertise to mobile developers in a powerful and easy-to-use package.”

Built By Google

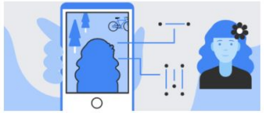
Optimized for Mobile

Easy to Use

Okay.. But what can it do?

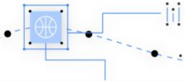
Vision APIs

Face Detection



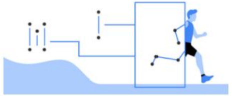
Selfie Segmentation

Barcode Scanning



Object Tracking

Pose Detection



Text Recognition

Digital Ink Recognition

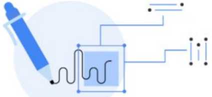
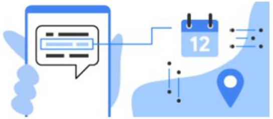


Image Labeling

Natural Language APIs

Language ID



Entity Extraction

Smart Reply

On-Device Translation

Who can use it??

Everyone!

ML Kit is available for both Android and iOS!

Meet our client, Steve

Steve is my Dad.

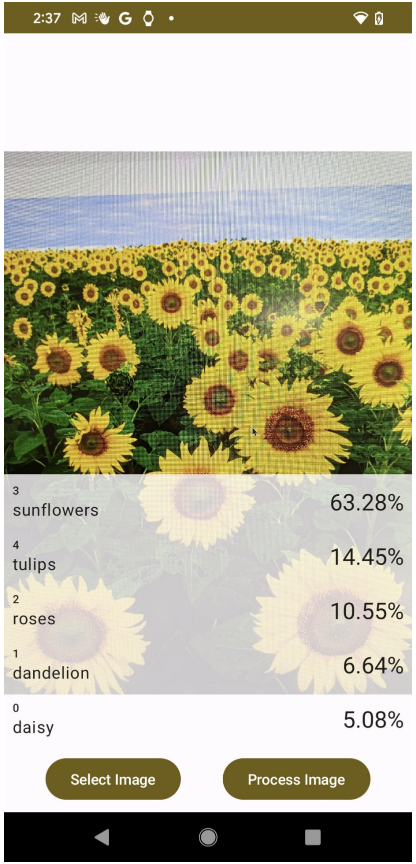
He really likes gardening.

He plants a lot of things and can't always remember what everything is.

He finally knows what I do for work.

With ML Kit, we can build an app to help Steve

Let’s get our app started!



3
sunflowers 63.28%

4
tulips 14.45%

2
roses 10.55%

1
dandelion 6.64%

0
daisy 5.08%

First, a little vocabulary + background

We'll be writing a native Android app using Jetpack Compose.

First, a little vocabulary + background

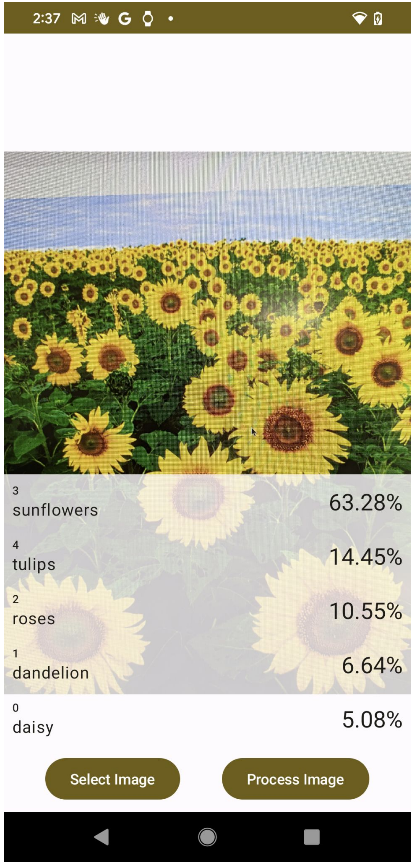
Jetpack Compose is Android's modern native UI Toolkit: declarative + all in Kotlin.

First, a little vocabulary + background

All of our code is will be in our single MainActivity.

What are we building?

We're going to build a simple single Activity app.



3
sunflowers 63.28%

4
tulips 14.45%

2
roses 10.55%

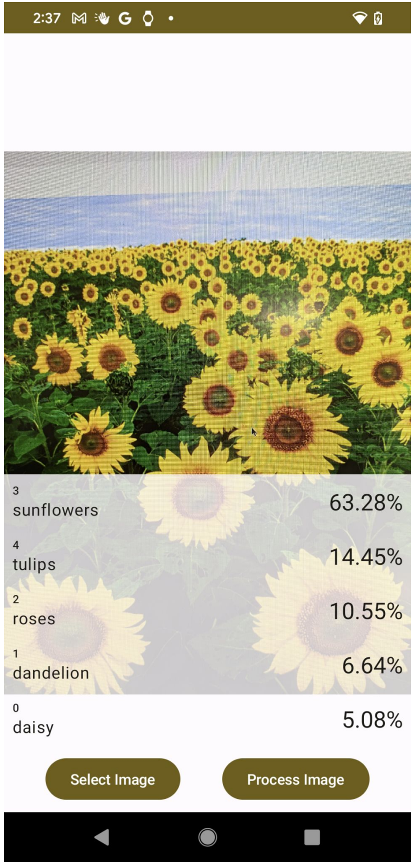
1
dandelion 6.64%

0
daisy 5.08%

What are we building?

We're going to build a simple single Activity app.

"Empty Compose Activity"



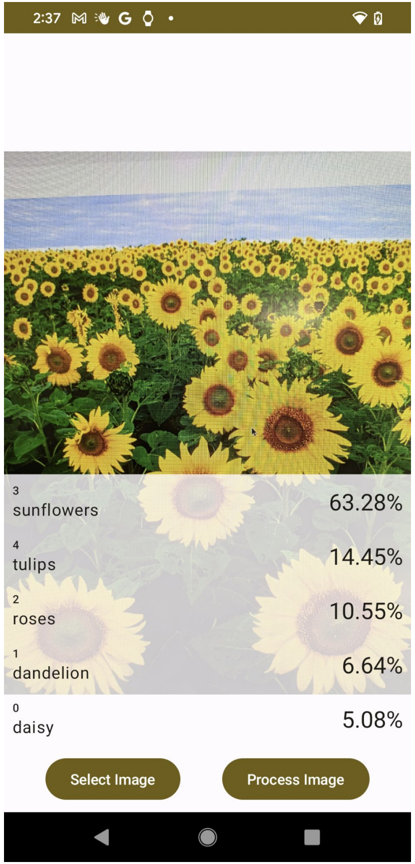
3 sunflowers	63.28%
4 tulips	14.45%
2 roses	10.55%
1 dandelion	6.64%
0 daisy	5.08%

What are we building?

We're going to build a simple single Activity app.

"Empty Compose Activity"

(gives us things like compose dependencies, basic theme, structure, …)



3 sunflowers	63.28%
4 tulips	14.45%
2 roses	10.55%
1 dandelion	6.64%
0 daisy	5.08%

Now we're going to switch over to Android Studio

We're going to get familiar with Android Studio

We're going to learn to run an Android App

Understand the structure of an Android codebase

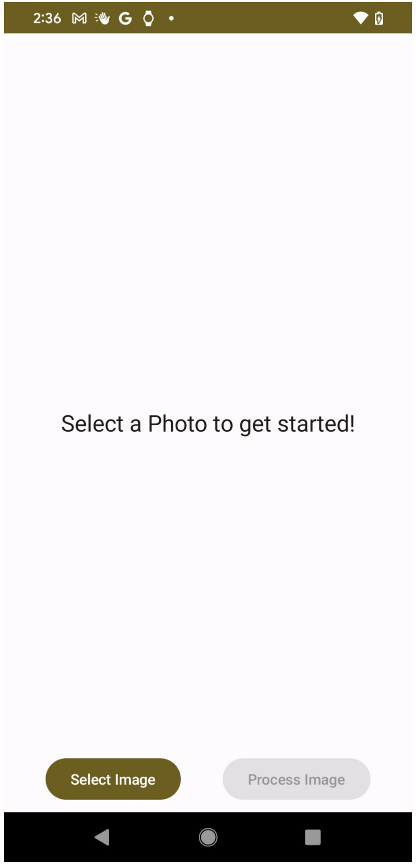
Learn the basics of Jetpack Compose

Start building our Plant App

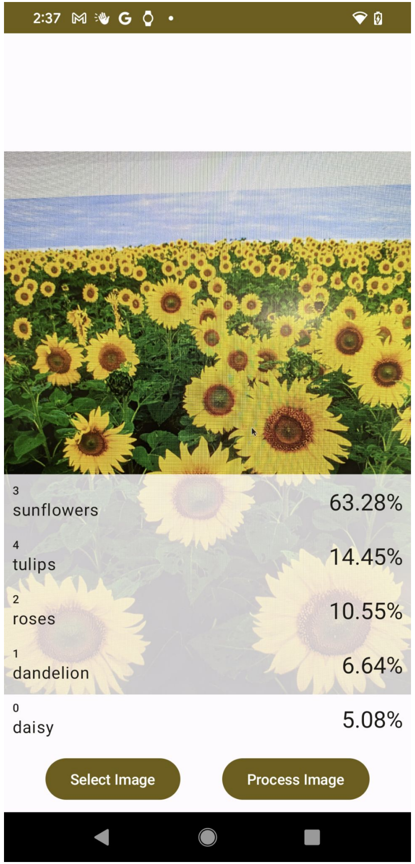
Part 1

Building the App UI

Start here



Select a Photo to get started!



3
sunflowers 63.28%

4
tulips 14.45%

2
roses 10.55%

1
dandelion 6.64%

0
daisy 5.08%

Helpful Links

Learn more about Kotlin

Kotlin Koans - Kotlin Koans are a series of exercises to get you familiar with the Kotlin syntax.

Introduction to programming in Kotlin - Learn introductory programming concepts in Kotlin to prepare for building Android apps in Kotlin.

Kotlin Fundamentals - Work through this set of codelabs to learn more about nullability, classes and objects, and lambda functions in Kotlin.

Kotlin Documentation - Get started with the first steps or jump to specific topics where you're most interested.

Helpful Links

Learn more about Compose

Compose Camp - Join Compose Camps events and other GDG meetups that are happening both in person and virtually around the world. Learn and connect with other developers and continue to grow your skill set in this hands-on format!

Android Basics with Compose - In this course, you'll learn the basics of building Android apps with Jetpack Compose, the new UI toolkit for building Android apps. Along the way, you'll develop a collection of apps to start your journey as an Android developer.

Jetpack Compose for Android Developers - Build on your knowledge from the Basics course and learn about more advanced topics like Architecture, Accessibility, Animations, Form Factors and more! (If you felt very comfortable with tonight's topics, feel free to start with this course!).

Get started with Jetpack Compose - The Compose docs have links to many specific topics if you just want to explore what you can do with Compose instead of working through a Codelab or Course. Explore topics like Performance, side-effects, Android Studio features and more that we didn't have time to cover tonight.

Part 1 Hints

Hints to get you started

Think about which parts of the screen need to change - these will need to be controlled with state variables (you'll need two state variables, all will be defined in `Content`)

Think about what how UI elements are oriented compared to others - think about if their stacked (Box), side by side (Row), or on top of each other (Column). Use Lazy Column / Row if needed. Use modifiers to add padding, style, behavior, etc. Use the Previews.

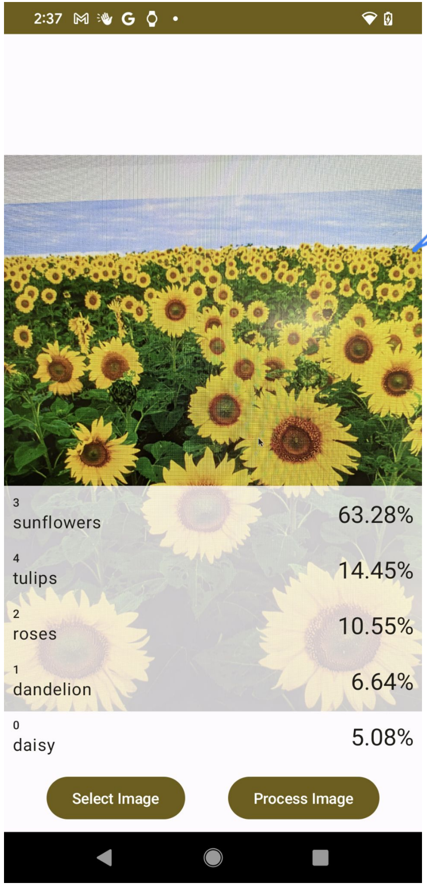
Don't worry about selecting the image just yet - we're just building the UI

Google and ask questions

What are we building?

Use Columns, Rows, and Boxes!

We'll use foundational composables to build the UI



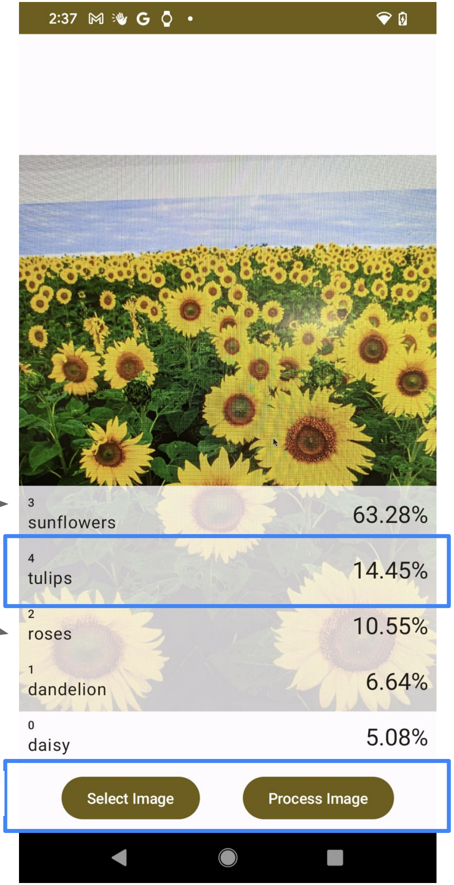
3 sunflowers	63.28%
4 tulips	14.45%
2 roses	10.55%
1 dandelion	6.64%
0 daisy	5.08%

Image(...)

LazyList(...)

Some Buttons(...)

What are we building?



```
3
sunflowers          63.28%
4
tulips              14.45%
2
roses               10.55%
1
dandelion            6.64%
0
daisy                5.08%
```

index

label

```
@Composable
private fun ImageContent(
    bitmap: Bitmap?
)
```

```
@Composable
private fun LabelRow(
    label: String,
    confidence: Float,
    index: Int,
)
```

```
@Composable
private fun ButtonRow(
    processButtonEnabled: Boolean,
    ...
)
```

Building the UI

```
@Composable
fun Content() {

   ...

   Box(modifier = Modifier.fillMaxSize()) {
       ImageContent(
           bitmap = bitmap,
       )
       Column(modifier = Modifier.align(Alignment.BottomCenter)) {
           LazyColumn(labels = labels){
               items(labels) { label -> ... }
           }
           ButtonRow(
               onSelectButtonClick = onSelectButtonClick,
               onProcessButtonClick = onProcessButtonClick
           )
       }
   }
}
```

If you’re stuck:

Looking for more hints:
go to 2-optional-main-activity-starter-hints

Check what you’re doing vs
3-main-activity-part-1-step-1

Part 1, Step 2

Now that we have our UI setup, we need to be able to select an image from our photos

To do this we'll use a Launcher for Activity Result, that lets us select a photo and use that information in our app.

Then convert the URI to a bitmap to use in our image function

We'll launch our launcher from our "Select Image" Button

Step One
Prepare the Input Image

First, we need to let Steve pick an Image from his photos.

```
@Composable
fun Content() {

    ...

    Box(
        modifier = Modifier.fillMaxSize()
     ) {
        ImageDisplay(bitmap)
        Column(
            modifier = Modifier.align(Alignment.BottomCenter)
        ) {
            LabelsColumn(labels = labels)
            ImageButtonRow(
                bitmap = bitmap,
                onSelectButtonClick = onSelectButtonClick,
                onProcessButtonClick = onProcessButtonClick
            )
        }
    }
}
```

Step One
Prepare the Input Image

First, we need to let Steve pick an Image from his photos.

```
@Composable
fun Content() {

    ...

    // instantiate launcher
    val context = LocalContext.current
    val launcher = rememberLauncherForActivityResult(
        contract = ActivityResultContracts.GetContent()
    ) { uri: Uri? ->
        // do something with the uri
    }

    Box(
        modifier = Modifier.fillMaxSize()
     ) {
        ImageDisplay(bitmap)
        Column(
            modifier = Modifier.align(Alignment.BottomCenter)
        ) {
            LabelsColumn(labels = labels)
            ImageButtonRow(
                bitmap = bitmap,
                onSelectButtonClick = onSelectButtonClick,
                onProcessButtonClick = onProcessButtonClick
            )
```

Step One
Prepare the Input Image

First, we need to let Steve pick an Image from his photos.

```
@Composable
fun Content() {
    // create state variables
    var bitmap by remember { mutableStateOf<Bitmap?>(null) }
    var labels by remember {
        mutableStateOf<List<String>>(emptyList())
    }

    val launcher = ...

    Box(
        modifier = Modifier.fillMaxSize()
     ) {
        ImageDisplay(bitmap)
        Column(
            modifier = Modifier.align(Alignment.BottomCenter)
        ) {
            LabelsColumn(labels = labels)
            ImageButtonRow(
                bitmap = bitmap,
                onSelectButtonClick = onSelectButtonClick,
                onProcessButtonClick = onProcessButtonClick
            )
        }
    }
}
```

Step One
Prepare the Input Image

First, we need to let Steve pick an Image from his photos.

Now, we need to take that URI and transform it into a Bitmap.

```
@Composable
fun Content() {

    var bitmap by remember { mutableStateOf<Bitmap?>(null) }
    var labels by remember {
        mutableStateOf<List<ImageLabel>>(emptyList())
    }

    val context = LocalContext.current
    val launcher = rememberLauncherForActivityResult(
        contract = ActivityResultContracts.GetContent()
    ) { uri: Uri? ->
        if (uri != null) {
           bitmap = ...
             // convert imageUri to bitmap?
        }
        labels = emptyList()
    }

    Box(
        modifier = Modifier.fillMaxSize()
     ) {
        ImageDisplay(bitmap)
        Column(
            modifier = Modifier.align(Alignment.BottomCenter)
        ) {
```

Step One
Prepare the Input Image

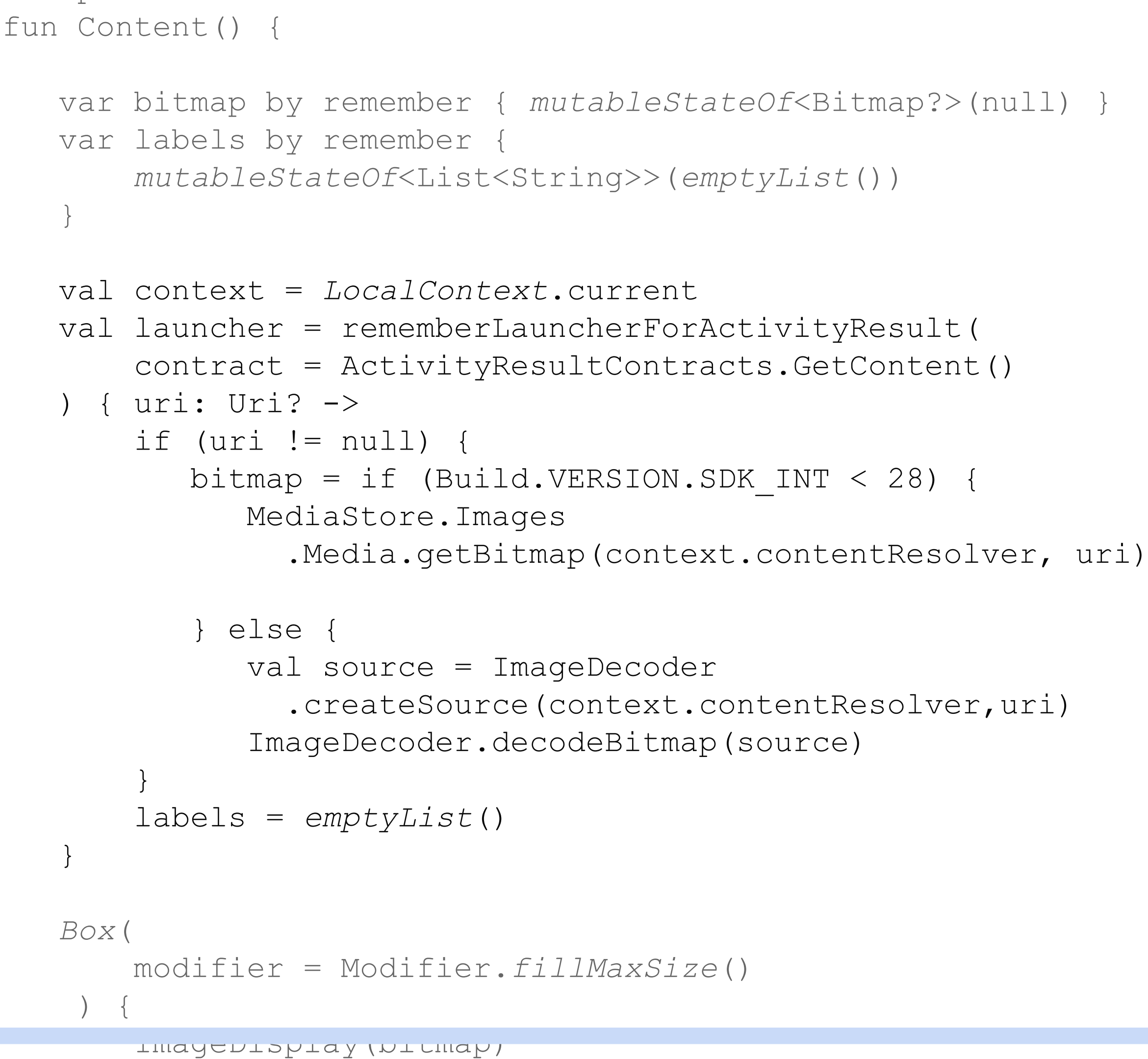
```
if (uri != null) {
    bitmap = if (Build.VERSION.SDK_INT < 28) {
        MediaStore.Images
            .Media.getBitmap(context.contentResolver, uri)

    } else {
        val source = ImageDecoder
            .createSource(context.contentResolver,uri)
        ImageDecoder.decodeBitmap(source)
    }
}
```

Step One
Prepare the Input Image

First, we need to let Steve pick an Image from his photos.

Now, we need to take that URI and transform it into a Bitmap.



```
fun Content() {

   var bitmap by remember { mutableStateOf<Bitmap?>(null) }
   var labels by remember {
       mutableStateOf<List<String>>(emptyList())
   }

   val context = LocalContext.current
   val launcher = rememberLauncherForActivityResult(
       contract = ActivityResultContracts.GetContent()
   ) { uri: Uri? ->
       if (uri != null) {
          bitmap = if (Build.VERSION.SDK_INT < 28) {
             MediaStore.Images
               .Media.getBitmap(context.contentResolver, uri)

          } else {
             val source = ImageDecoder
               .createSource(context.contentResolver,uri)
             ImageDecoder.decodeBitmap(source)
       }
       labels = emptyList()
   }

   Box(
       modifier = Modifier.fillMaxSize()
    ) {
       ImageDisplay(bitmap)
```

Step One
Prepare the Input Image

```
@Composable
fun Content() {
   var bitmap by remember { mutableStateOf<Bitmap?>(null) }
   ...
   val launcher = rememberLauncherForActivityResult(...) {
       bitmap = uri?.let { imageUri ->
          getBitmapFromUri(context.contentResolver, imageUri)
       }
       labels = emptyList()
   }

   Box {
       ImageContent(bitmap)
       ...
   }
}


@Composable
private fun ImageContent(
    bitmap: Bitmap?
) {
    if (bitmap != null) {
        Image(
            bitmap = bitmap.asImageBitmap(),
            contentDescription = "Selected Photo",
            modifier = Modifier
                 fillMaxSize()
```

Step One
Prepare the Input Image

```
    Box {
        ImageContent(bitmap)
        ...
    }
}


@Composable
private fun ImageContent(
    bitmap: Bitmap?
) {
    if (bitmap != null) {
        Image(
            bitmap = bitmap.asImageBitmap(),
            contentDescription = "Selected Photo",
            modifier = Modifier
                .fillMaxSize()
                .wrapContentSize()
        )
    } else {
        Text(
            text = "Select a Photo to get started!",
            style = MaterialTheme.typography.titleLarge,
            modifier = Modifier
                .fillMaxSize()
                .wrapContentSize()
        )
    }
}
```

Step One
Prepare the Input Image

Finally we need to tie these all together with the UI

```
    mutableStateOf<List<ImageLabel>>(emptyList())
}

val context = LocalContext.current
val launcher = rememberLauncherForActivityResult(
    contract = ActivityResultContracts.GetContent()
) { uri: Uri? ->
    bitmap = uri?.let { imageUri ->
       getBitmapFromUri(context.contentResolver, imageUri)
    }
    labels = emptyList()
}

// create and use select lambda function
val onSelectButtonClick = { launcher.launch("image/*") }
val onProcessButtonClick = { /* TODO */ }

Box {
    ...
    Column {
        ...
        ImageButtonRow(
            bitmap = bitmap,
            onSelectButtonClick = onSelectButtonClick,
            onProcessButtonClick = onProcessButtonClick
        )
    }
}
}
```

If you’re stuck:

If stuck with the launcher - code is available here

This code is completed in
4-main-activity-end-part-1

At this point, you should be able to run your app,
select a photo, and that photo will show in the app.

Part 2 -
Introducing ML Kit

ML Kit Image Labeling Docs

Default Model Map

Android Setup Docs

First, a little vocabulary + background

Now that we have an app, we can talk about setting up ML Kit.

How do we use it?

We need to add the Image Labeling dependency to our gradle

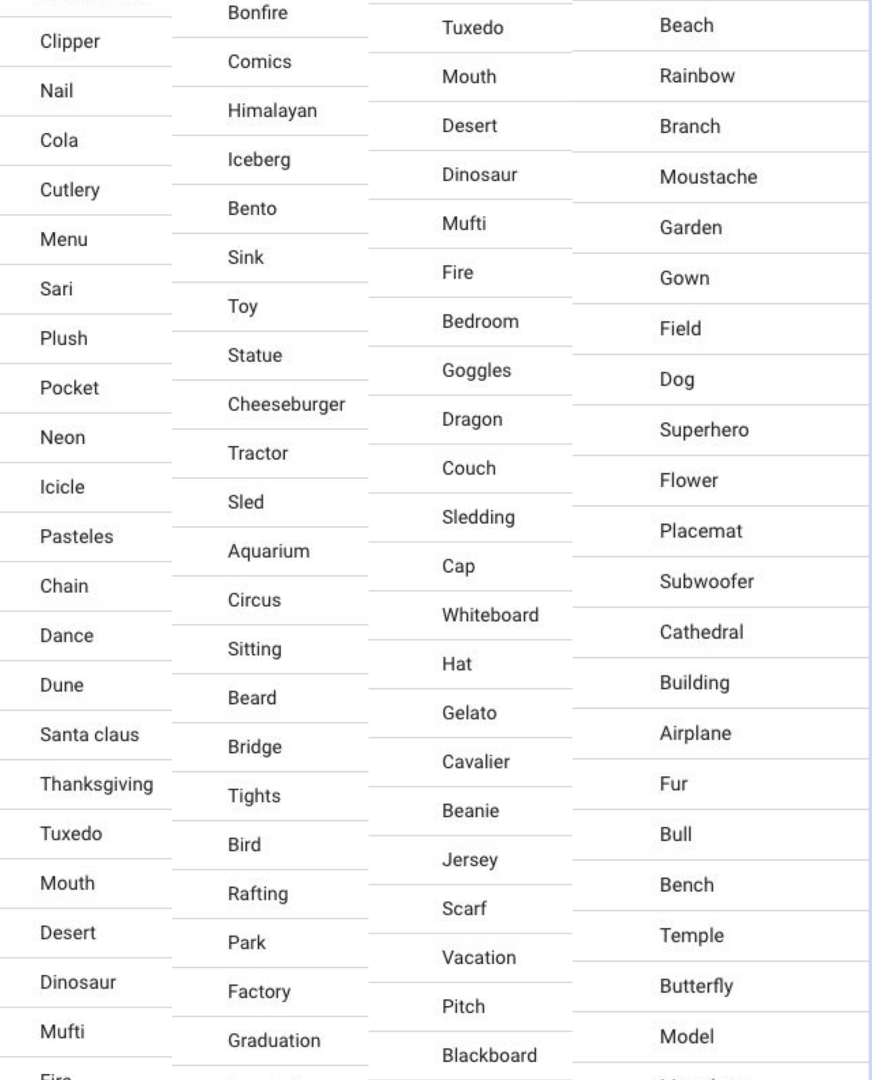
```
dependencies {
  ...

  // Use this dependency to bundle the pipeline with your app
  implementation 'com.google.mlkit:image-labeling:17.0.7'
}
```

We're going to bundle it with the app

```
dependencies {
  ...

  // Use this dependency to use the dynamically downloaded model in Google Play Services
  implementation 'com.google.android.gms:play-services-mlkit-image-labeling:16.0.8'
}
```



Clipper
Nail
Cola
Cutlery
Menu
Sari
Plush
Pocket
Neon
Icicle
Pasteles
Chain
Dance
Dune
Santa claus
Thanksgiving
Tuxedo
Mouth
Desert
Dinosaur
Mufti
Fire

Bonfire
Comics
Himalayan
Iceberg
Bento
Sink
Toy
Statue
Cheeseburger
Tractor
Sled
Aquarium
Circus
Sitting
Beard
Bridge
Tights
Bird
Rafting
Park
Factory
Graduation

Tuxedo
Mouth
Desert
Dinosaur
Mufti
Fire
Bedroom
Goggles
Dragon
Couch
Sledding
Cap
Whiteboard
Hat
Gelato
Cavalier
Beanie
Jersey
Scarf
Vacation
Pitch
Blackboard

Beach
Rainbow
Branch
Moustache
Garden
Gown
Field
Dog
Superhero
Flower
Placemat
Subwoofer
Cathedral
Building
Airplane
Fur
Bull
Bench
Temple
Butterfly
Model

We’re just going to use the Base Model

Step Two
Process the Input Image

We’re going to update our labels from String to ImageLabel

```
@Composable
fun Content() {
   // update our state variable
   var bitmap by remember { mutableStateOf<Bitmap?>(null) }
   var labels by remember {
       mutableStateOf<List<ImageLabel>>(emptyList())
   }

   val launcher = ...

   Box(
       modifier = Modifier.fillMaxSize()
    ) {
       ImageDisplay(bitmap)
       Column(
           modifier = Modifier.align(Alignment.BottomCenter)
       ) {
           LabelsColumn(labels = labels)
           ImageButtonRow(
               bitmap = bitmap,
               onSelectButtonClick = onSelectButtonClick,
               onProcessButtonClick = onProcessButtonClick
           )
       }
   }
}
```

Step Two
Process the Input Image

Each ImageLabel object has a label field, confidence field, and index field

```
@Composable
fun Content() {
    // update our state variable
    var bitmap by remember { mutableStateOf<Bitmap?>(null) }
    var labels by remember {
        mutableStateOf<List<ImageLabel>>(emptyList())
    }

    val launcher = ...

    Box(
        modifier = Modifier.fillMaxSize()
     ) {
        ImageDisplay(bitmap)
        Column(
            modifier = Modifier.align(Alignment.BottomCenter)
        ) {
            LabelsColumn(labels = labels)
            ImageButtonRow(
                bitmap = bitmap,
                onSelectButtonClick = onSelectButtonClick,
                onProcessButtonClick = onProcessButtonClick
            )
        }
    }
}
```

Step Two
Process the Input Image

We can now display the image but we haven't used ML Kit yet

```
           mutableStateOf<List<ImageLabel>>(emptyList())
   }

   val context = LocalContext.current
   val launcher = rememberLauncherForActivityResult(
       contract = ActivityResultContracts.GetContent()
   ) { uri: Uri? ->
       bitmap = uri?.let { imageUri ->
          getBitmapFromUri(context.contentResolver, imageUri)
       }
       labels = emptyList()
   }

   val onSelectButtonClick = { launcher.launch("image/*") }
   // create and use the process lambda function
   val onProcessButtonClick = { /* TODO */ }

   Box {
       ...
       Column {
           ...
           ImageButtonRow(
               bitmap = bitmap,
               onSelectButtonClick = onSelectButtonClick,
               onProcessButtonClick = onProcessButtonClick
           )
       }
   }
}
```

Create our Labeler

```
private val labeler =
   ImageLabeling
   .getClient(
     ImageLabelerOptions.DEFAULT_OPTIONS
   )
```

Process the image

```
private fun process(



) {




}
```

Process the image

```
private fun process(
   bitmap: Bitmap?,



) {
   if (bitmap == null) return ???




}
```

Process the image

```
private fun process(
    bitmap: Bitmap?,
    imageLabeler: ImageLabeler,
    rotation: Int = 0,

) {
    if (bitmap == null) return ???

    labeler.process(bitmap, rotation)
         .addOnSuccessListener { labels -> ??? }
         .addOnFailureListener { ??? }
}
```

Process the image

```
private fun process(
    bitmap: Bitmap?,
    imageLabeler: ImageLabeler,
    rotation: Int = 0,
    onComplete: (List<ImageLabel>) -> Unit
) {
    if (bitmap == null) return onComplete(emptyList())

    labeler.process(bitmap, rotation)
         .addOnSuccessListener { labels -> onComplete(labels) }
         .addOnFailureListener { onComplete(emptyList()) }
}
```

Step Two
Process the Input Image

We can now display the image but we haven’t used ML Kit yet

```
       mutableStateOf<List<ImageLabel>>(emptyList())
   }

   val context = LocalContext.current
   val launcher = rememberLauncherForActivityResult(
       contract = ActivityResultContracts.GetContent()
   ) { uri: Uri? ->
       bitmap = uri?.let { imageUri ->
          getBitmapFromUri(context.contentResolver, imageUri)
       }
       labels = emptyList()
   }

   val onSelectButtonClick = { launcher.launch("image/*") }
   // create and use the process lambda function
   val onProcessButtonClick = { /* TODO */ }

   Box {
       ...
       Column {
           ...
           ImageButtonRow(
               bitmap = bitmap,
               onSelectButtonClick = onSelectButtonClick,
               onProcessButtonClick = onProcessButtonClick
           )
       }
   }
}
```

Step Two
Process the Input Image

We can now display the image but we haven’t used ML Kit yet

```
var bitmap by remember { mutableStateOf<Bitmap?>(null) }
var labels by remember {
    mutableStateOf<List<ImageLabel>>(emptyList())
}
...

val onSelectButtonClick = { launcher.launch("image/*") }
// create and use the process lambda function
val onProcessButtonClick = {
     processDefault(
         bitmap = bitmap,
         imageLabeler = labeler,
         onComplete = { outputLabels ->
              labels = outputLabels
         }
     )
}

Box {
    ...
    Column {
        ...
        ImageButtonRow(
            bitmap = bitmap,
            onSelectButtonClick = onSelectButtonClick,
            onProcessButtonClick = onProcessButtonClick
        )
    }
```

Step Three
Display the labels

Finally, now that we have the labels from our processed image, we should display those

```
@Composable
fun Content() {

    ...

    Box(
        modifier = Modifier.fillMaxSize()
    ) {
        ImageDisplay(bitmap)
        Column(
            modifier = Modifier.align(Alignment.BottomCenter)
        ) {
            LabelsColumn(labels = labels)
            ImageButtonRow(
                bitmap = bitmap,
                onSelectButtonClick = onSelectButtonClick,
                onProcessButtonClick = onProcessButtonClick
            )
        }
    }
}
```

If you’re stuck:

This code is completed in
5-main-activity-part-2

At this point, you should be able to run your app, select a photo, and then process the image using the default model. The resulting labels should show in the app.

And with that we have an app!

Let’s try it out!

Where to go from here

Maybe make a custom data model to label plants

Let’s do it!

Part 3 - Custom Models

Custom Models on Android Setup Docs

How do we get started with custom models?

The docs for Custom Models with ML Kit tell you there are four different ways to get started:

- TensorFlow Hub
- TensorFlow
- Vertex AI
- MediaPipe Model Maker

What are the differences??

How do we get started with custom models?

The docs for Custom Models with ML Kit tells you there are four different ways to get started:

- TensorFlow Hub
- TensorFlow
- Vertex AI
- MediaPipe Model Maker

offers a wide range of pre-trained image classification models

How do we get started with custom models?

The docs for Custom Models with ML Kit tells you there are four different ways to get started:

- TensorFlow Hub
- TensorFlow
- Vertex AI
- MediaPipe Model Maker

Input:

- RGB format
- Int8 or float32
- [batch = 1, height, width, channel = 3]

Output:

2d or 4d tensor

How do we get started with custom models?

The docs for Custom Models with ML Kit tells you there are four different ways to get started:

- TensorFlow Hub
- TensorFlow
- Vertex AI
- MediaPipe Model Maker

train a model with TensorFlow and then convert it to TensorFlow Lite

How do we get started with custom models?

The docs for Custom Models with ML Kit tells you there are four different ways to get started:

- TensorFlow Hub
- TensorFlow
- Vertex AI
- MediaPipe Model Maker

AI platform built on Google Cloud For all building, deploying, and scaling ML

How do we get started with custom models?

The docs for Custom Models with ML Kit tells you there are four different ways to get started:

- TensorFlow Hub
- TensorFlow
- AutoML
- MediaPipe Model Maker

Re-train a model (transfer learning), takes less time and requires less data than training a model from scratch

To get started:

We'll be using the Image Classification Model Customization guide and Google Colab

[Customization Guide]

[Google Colab]

Before you get started

There are a lot of defaults provided for you. Experiment with these! Be sure to check out Model Tuning to see the options and parameters you can tweak.

You can read through the image classification overview for the different models available

This is an early release and they plan to release a new model maker colab (currently broken) built on MediaPipe soon.

Creating a Custom Model

To install the libraries for customizing a model, run the following commands:

```
!python --version Note: add `!` - this is missing in the guide
!pip install --upgrade pip
!pip install mediapipe-model-maker
```

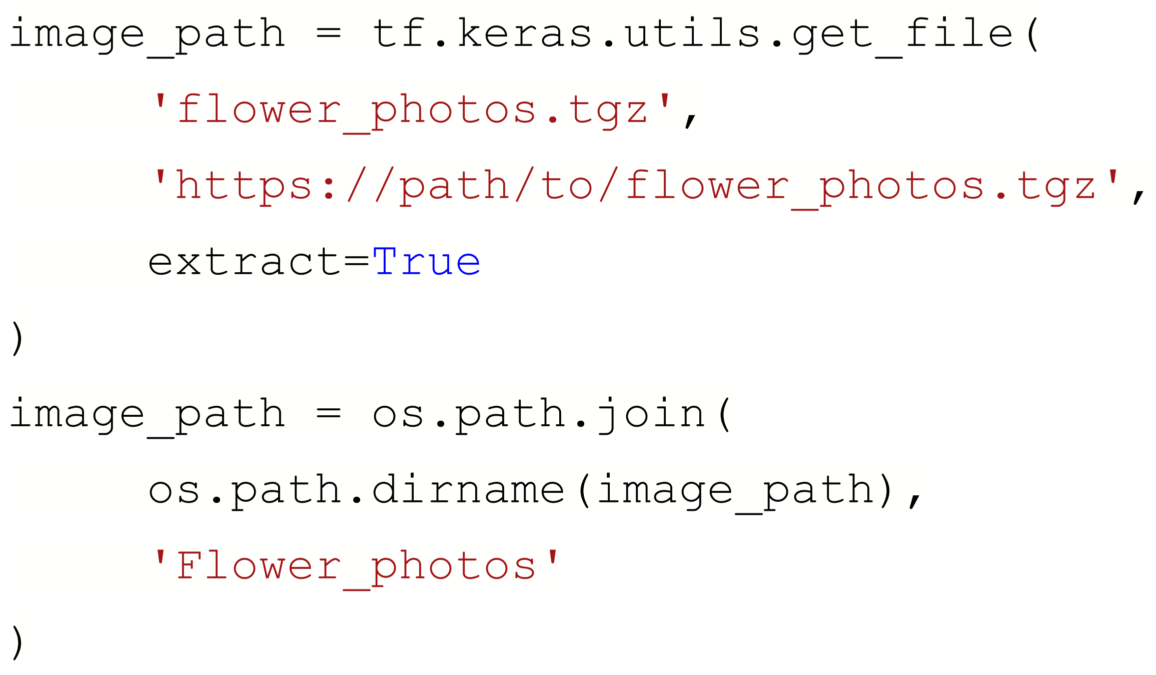
Use the following code to import the required Python classes:

```
from google.colab import files
import os
import tensorflow as tf
assert tf.__version__.startswith('2')

from mediapipe_model_maker import image_classifier

import matplotlib.pyplot as plt
```

Creating a Custom Model



```
image_path = tf.keras.utils.get_file(
     'flower_photos.tgz',
     'https://path/to/flower_photos.tgz',
     extract=True
)
image_path = os.path.join(
     os.path.dirname(image_path),
     'Flower_photos'
)
```

Review the data

```
print(image_path)
labels = []
for i in os.listdir(image_path):
  if os.path.isdir(os.path.join(image_path, i)):
    labels.append(i)
print(labels)

NUM_EXAMPLES = 5

for label in labels:
  label_dir = os.path.join(image_path, label)
  example_filenames = os.listdir(label_dir)[:NUM_EXAMPLES]
  fig, axs = plt.subplots(1, NUM_EXAMPLES, figsize=(10,2))
  for i in range(NUM_EXAMPLES):
    axs[i].imshow(plt.imread(os.path.join(label_dir, example_filenames[i])))
    axs[i].get_xaxis().set_visible(False)
    axs[i].get_yaxis().set_visible(False)
  fig.suptitle(f'Showing {NUM_EXAMPLES} examples for {label}')

plt.show()
```

Create the dataset

```
data = image_classifier.Dataset.from_folder(image_path)
train_data, remaining_data = data.split(0.8)
test_data, validation_data = remaining_data.split(0.5)
```

Retrain your model

```
spec = image_classifier.SupportedModels.MOBILENET_V2
hparams = image_classifier.HParams(export_dir="exported_model")
options = image_classifier.ImageClassifierOptions(supported_model=spec,
hparams=hparams)


model = image_classifier.ImageClassifier.create(
    train_data = train_data,
    validation_data = validation_data,
    options=options,
)
```

Creating a Custom Model

```
loss, acc = model.evaluate(test_data)
print(f'Test loss:{loss}, Test accuracy:{acc}')

model.export_model()
```

Test and Export

```
model image_classifier
    .ImageClassifier
    .create(train_data, validation_data, options)
```

Learn more about transfer learning here

Creating a Custom Model

```
@classmethod
tflite_model_maker.image_classifier.create(
    train_data,
    model_spec='efficientnet_lite0',
    validation_data=None,
    batch_size=None,
    epochs=None,
    steps_per_epoch=None,
    train_whole_model=None,
    dropout_rate=None,
    learning_rate=None,
    momentum=None,
    shuffle=False,
    use_augmentation=False,
    use_hub_library=True,
    warmup_steps=None,
    model_dir=None,
    do_train=True
)
```

Creating a Custom Model

```
@classmethod
tflite_model_maker.image_classifier.create(
    train_data,
    model_spec='efficientnet_lite0',
    validation_data=None,
    batch_size=None,
    epochs=None,
    steps_per_epoch=None,
    train_whole_model=None,
    dropout_rate=None,
    learning_rate=None,
    momentum=None,
    shuffle=False,
    use_augmentation=False,
    use_hub_library=True,
    warmup_steps=None,
    model_dir=None,
    do_train=True
)
```

Creating a Custom Model

```
@classmethod
tflite_model_maker.image_classifier.create(
    train_data,
    model_spec='efficientnet_lite0',
    validation_data=None,
    batch_size=None,
    epochs=None,
    steps_per_epoch=None,
    train_whole_model=None,
    dropout_rate=None,
    learning_rate=None,
    momentum=None,
    shuffle=False,
    use_augmentation=False,
    use_hub_library=True,
    warmup_steps=None,
    model_dir=None,
    do_train=True
)
```

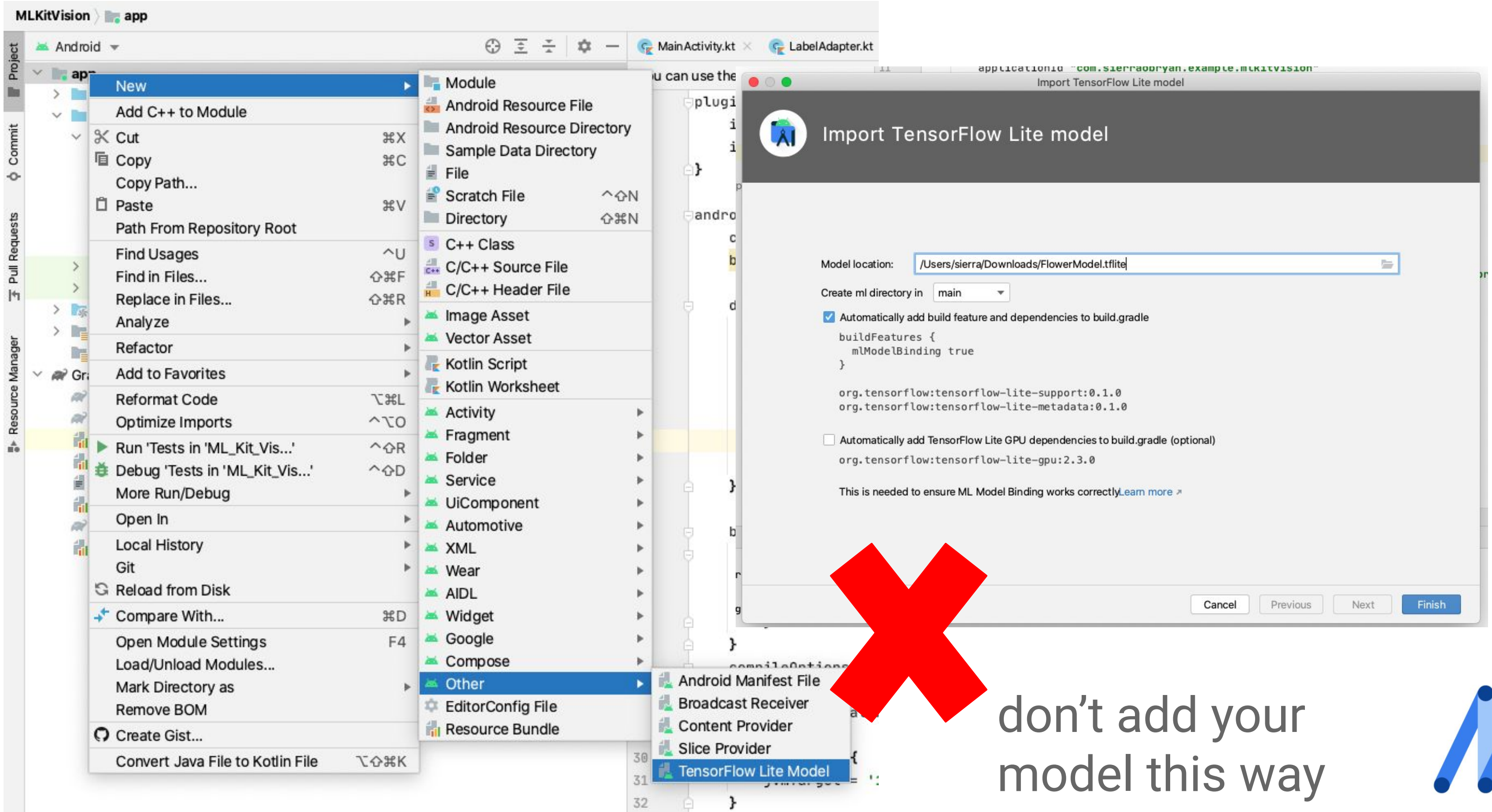
Add custom model dependency

```
dependencies {
  ...

  // Use this dependency to bundle the pipeline with your app
  implementation 'com.google.mlkit:image-labeling-custom:17.0.1'
}
```

and add the custom model to assets folder

`app > assets > customModel.tflite`



Import TensorFlow Lite model

Model location: /Users/sierra/Downloads/FlowerModel.tflite

Create ml directory in: main

☑ Automatically add build feature and dependencies to build.gradle

```
buildFeatures {
  mlModelBinding true
}

org.tensorflow:tensorflow-lite-support:0.1.0
org.tensorflow:tensorflow-lite-metadata:0.1.0
```

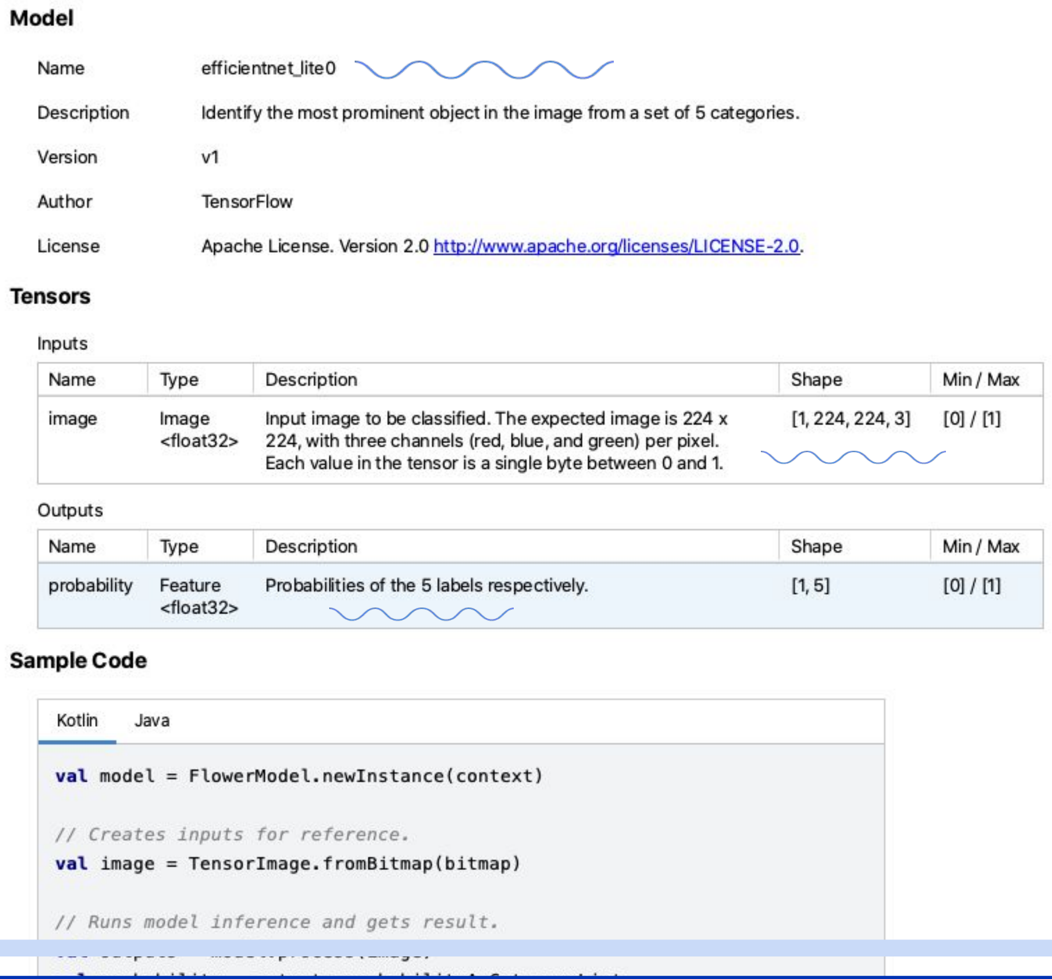
□ Automatically add TensorFlow Lite GPU dependencies to build.gradle (optional)

```
org.tensorflow:tensorflow-lite-gpu:2.3.0
```

This is needed to ensure ML Model Binding works correctlyLearn more ↗

don’t add your model this way

Adding the model to your project



Model

Name	efficientnet_lite0
Description	Identify the most prominent object in the image from a set of 5 categories.
Version	v1
Author	TensorFlow
License	Apache License. Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.

Tensors

Inputs

Name	Type	Description	Shape	Min / Max
image	Image <float32>	Input image to be classified. The expected image is 224 x 224, with three channels (red, blue, and green) per pixel. Each value in the tensor is a single byte between 0 and 1.	[1, 224, 224, 3]	[0] / [1]

Outputs

Name	Type	Description	Shape	Min / Max
probability	Feature <float32>	Probabilities of the 5 labels respectively.	[1, 5]	[0] / [1]

Sample Code

Kotlin Java

```
val model = FlowerModel.newInstance(context)

// Creates inputs for reference.
val image = TensorImage.fromBitmap(bitmap)

// Runs model inference and gets result.
```

Let’s use our new model!

```
// build the custom model
val customModel = LocalModel.Builder()
    .setAssetFilePath("model_media_pipe.tflite")
    .build()
```

Let’s use our new model!

```
// build the custom model
val customModel = LocalModel.Builder()
    .setAssetFilePath("model_media_pipe.tflite")
    .build()

// create our custom labeler options
val customImageLabelerOptions =
    CustomImageLabelerOptions.Builder(customModel)
         .build()
```

Let’s use our new model!

```
// build the custom model
val customModel = LocalModel.Builder()
    .setAssetFilePath("model_media_pipe.tflite")
    .build()

// create our custom labeler options
val customImageLabelerOptions =
    CustomImageLabelerOptions.Builder(customModel)
        .build()

// make the custom labeler
val customLabeler = ImageLabeling.getClient(customImageLabelerOptions)
```

Let’s use our new model!

```
val onProcessButtonClick = {
     processDefault(
         bitmap = bitmap,
         imageLabeler = customLabeler,
         onComplete = { outputLabels ->
              labels = outputLabels
         }
     )
}
```

And with that we have an app with a custom model!

Let’s try it out!

And on our original backyard image

Let's try it out!

YAY! We can label five flowers

Daisies

Dandelion

Roses

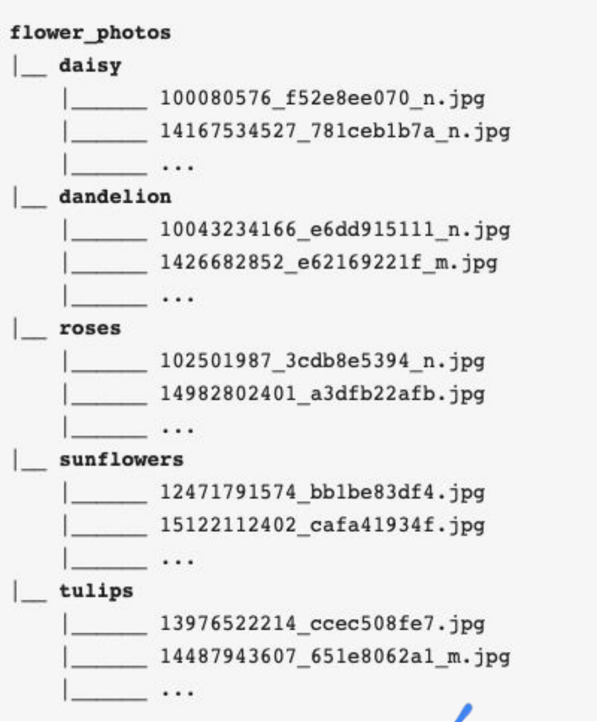
Sunflowers

Tulips

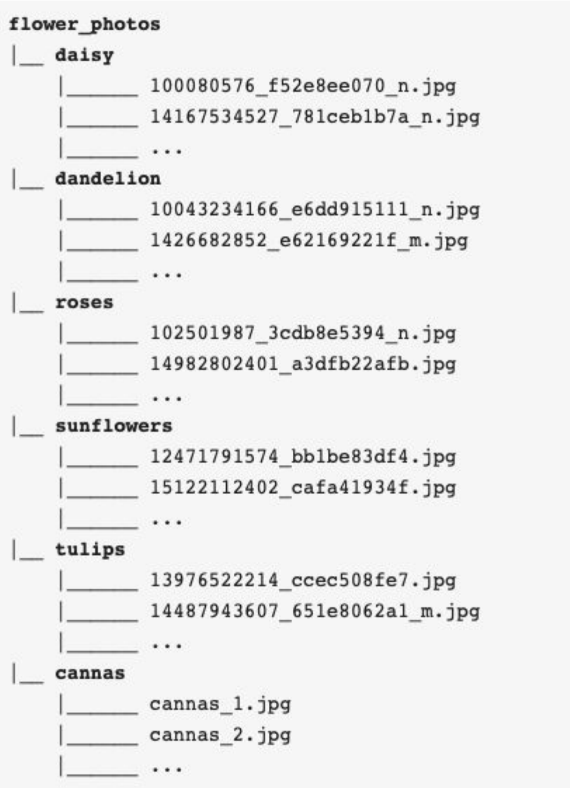
Cannas

But what happens if we want to label a sixth?

Easy - we’re ML Kit experts now

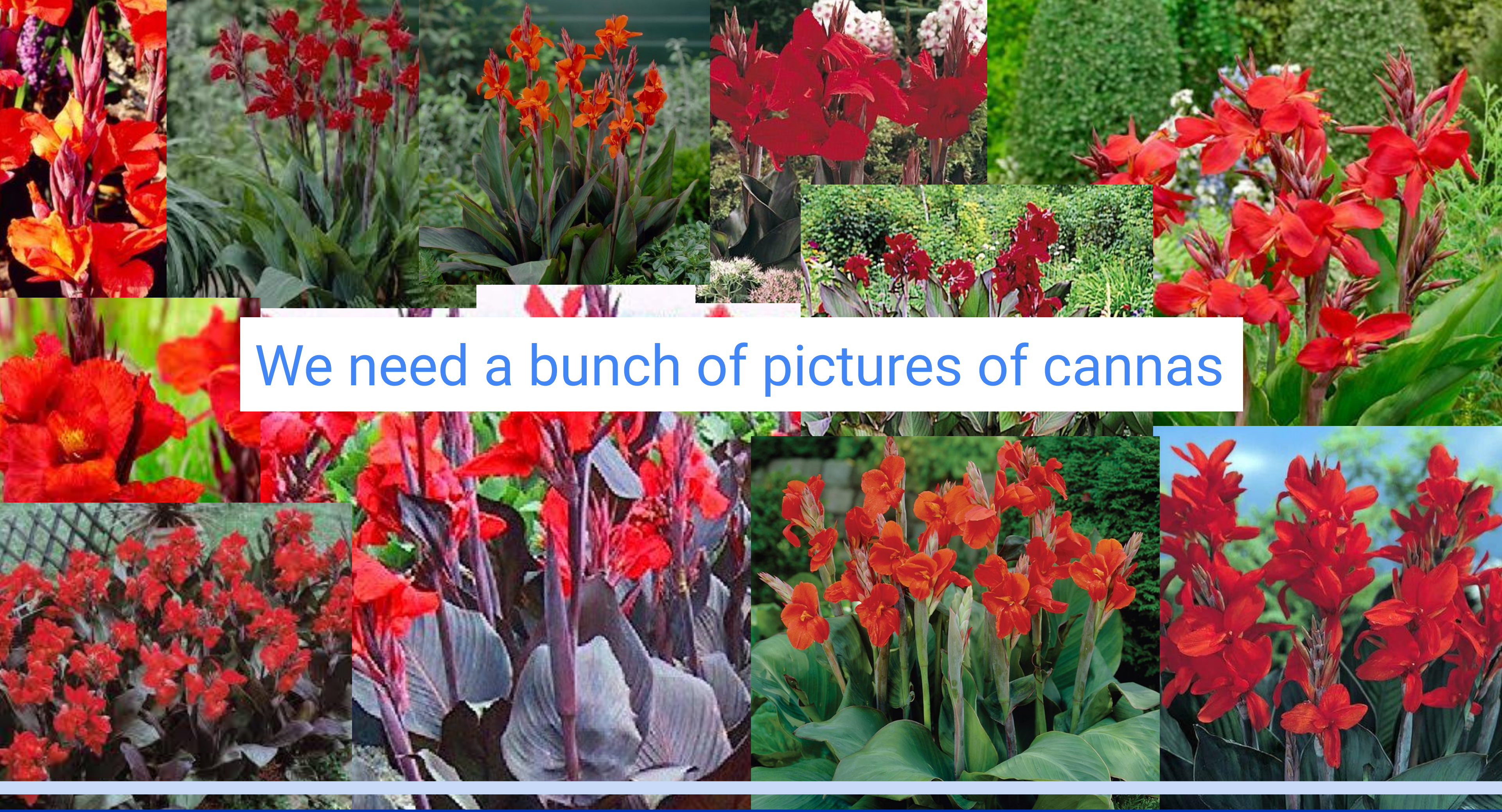


```
flower_photos
|__ daisy
    |______ 100080576_f52e8ee070_n.jpg
    |______ 14167534527_781ceb1b7a_n.jpg
    |______ ...
|__ dandelion
    |______ 10043234166_e6dd915111_n.jpg
    |______ 1426682852_e62169221f_m.jpg
    |______ ...
|__ roses
    |______ 102501987_3cdb8e5394_n.jpg
    |______ 14982802401_a3dfb22afb.jpg
    |______ ...
|__ sunflowers
    |______ 12471791574_bb1be83df4.jpg
    |______ 15122112402_cafa41934f.jpg
    |______ ...
|__ tulips
    |______ 13976522214_ccec508fe7.jpg
    |______ 14487943607_651e8062a1_m.jpg
    |______ ...
```



```
flower_photos
|__ daisy
    |______ 100080576_f52e8ee070_n.jpg
    |______ 14167534527_781ceb1b7a_n.jpg
    |______ ...
|__ dandelion
    |______ 10043234166_e6dd915111_n.jpg
    |______ 1426682852_e62169221f_m.jpg
    |______ ...
|__ roses
    |______ 102501987_3cdb8e5394_n.jpg
    |______ 14982802401_a3dfb22afb.jpg
    |______ ...
|__ sunflowers
    |______ 12471791574_bb1be83df4.jpg
    |______ 15122112402_cafa41934f.jpg
    |______ ...
|__ tulips
    |______ 13976522214_ccec508fe7.jpg
    |______ 14487943607_651e8062a1_m.jpg
    |______ ...
|__ cannas
    |______ cannas_1.jpg
    |______ cannas_2.jpg
    |______ ...
```

We need a bunch of pictures of cannas



Adding the model to your project

```
image_path = tf.keras.utils.get_file(
          'flowers-new.zip',
          'file:///content/flowers-new.zip',
          extract=True)
image_path = os.path.join(
          os.path.dirname(image_path),
          'Flowers-new')
```

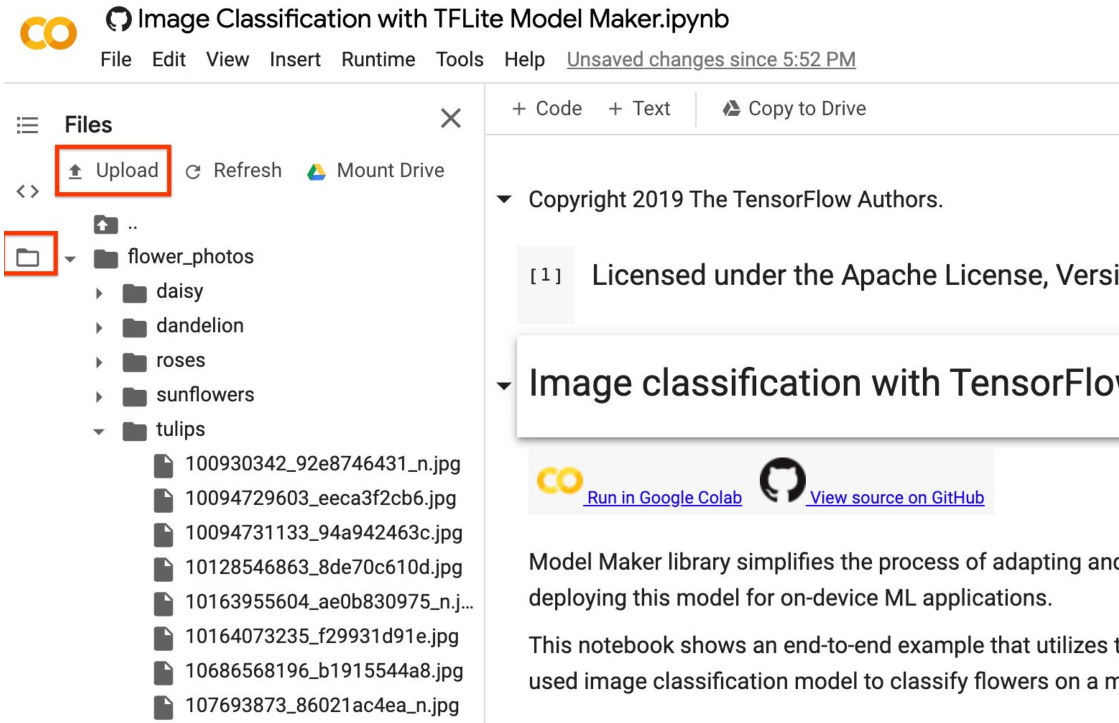


Image Classification with TFLite Model Maker.ipynb

File
Edit
View
Insert
Runtime
Tools
Help
Unsaved changes since 5:52 PM
+ Code
+ Text
Copy to Drive
Files
Upload
Refresh
Mount Drive
..
flower_photos
daisy
dandelion
roses
sunflowers
tulips
100930342_92e8746431_n.jpg
10094729603_eeca3f2cb6.jpg
10094731133_94a942463c.jpg
10128546863_8de70c610d.jpg
10163955604_ae0b830975_n.j...
10164073235_f29931d91e.jpg
10686568196_b1915544a8.jpg
107693873_86021ac4ea_n.jpg
Run in Google Colab
View source on GitHub

Copyright 2019 The TensorFlow Authors.

[1] Licensed under the Apache License, Versi

Image classification with TensorFlo

Model Maker library simplifies the process of adapting and deploying this model for on-device ML applications.

This notebook shows an end-to-end example that utilizes t used image classification model to classify flowers on a m

And with that we have an app with our custom model!

Let’s try it out!

Part 4 - Real Time Camera

Where to go from here

~~Maybe make a custom data model to label plants~~

Maybe use the camera so that I don’t have to load an image

Let’s do it!

warning: working with cameraX is … a lot

(we’ll walk through it together)

How do we get started with the camera?

Well actually this turns out to be a pretty frustrating and in depth process but here are the basics:

- You’ll need to add the camera dependencies to your gradle for the camera, lifecycle, and view
- You’ll also need to add the permission to your manifest and ask the user for permission
- Then we’ll add some code to the activity to process the image

How do we get started with the camera?

Well actually this turns out to be a pretty frustrating and in depth process but here are the basics (cont.) :

- We’ll use the ProcessCameraProvider and the ImageAnalysis classes to bind the camera to our lifecycle and build an image analyzer that will convert the imageProxy to a bitmap that can then be passed into the process function of our model and then back into our Label Column Composable (and display in the PreviewView)

Add Dependencies

```
dependencies {
  ...

  // CameraX Lifecycle library
  implementation "androidx.camera:camera-lifecycle:1.2.3"
  implementation "androidx.camera:camera-camera2:1.2.3"
  implementation "androidx.camera:camera-view:1.2.3"
}
```

Update the UI

```
@Composable
fun CameraContent() {

    var labels by remember {
        mutableStateOf<List<Category>>(emptyList())
    }

    Box(modifier = Modifier.fillMaxSize()) {

        CameraPreview { labels = it }

        LabelsColumn(labels = labels)
    }
}
```

Update the UI

```
@Composable
fun CameraContent() {
    // list of labels as state
    var labels by remember {
        mutableStateOf<List<Category>>(emptyList())
    }

    Box(modifier = Modifier.fillMaxSize()) {

        // live camera preview composable
        CameraPreview { labels = it }

        // label column composable
        LabelsColumn(labels = labels)
    }
}
```

Add permissions

```
// AndroidManifest.xml

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

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

    ...
</manifest>
```

Add permissions

```
/** Request permissions prior to launching exercise.**/
val permissionLauncher = rememberLauncherForActivityResult(
    ActivityResultContracts.RequestMultiplePermissions()
) { result ->
    if (result.all { it.value }) {
        Log.d("MyApp", "All required permissions granted")
    }
}

LaunchedEffect(Unit) {
    launch {
        permissionLauncher.launch(
            arrayOf(Manifest.permission.CAMERA)
        )
    }
}
```

Set up the Camera Preview Composable

```
@Composable
fun CameraPreview(
    onImageAnalyzerOutputs: (List<Category>) -> Unit
) {

}
```

Set up the Camera Preview Composable

```
@Composable
fun CameraPreview(
    onImageAnalyzerOutputs: (List<Category>) -> Unit
) {
    val lifecycleOwner = LocalLifecycleOwner.current
    val context = LocalContext.current
    val cameraProviderFuture = remember { ProcessCameraProvider.getInstance(context) }

    AndroidView(
        factory = { ctx ->... },
        modifier = Modifier.fillMaxSize(),
    )
}
```

```
val context = LocalContext.current
val cameraProviderFuture = remember { ProcessCameraProvider.getInstance(context)

AndroidView(
    factory = { ctx ->
        val previewView = PreviewView(ctx)
        val executor = ContextCompat.getMainExecutor(ctx)
        cameraProviderFuture.addListener({
            val cameraProvider = cameraProviderFuture.get()
            val preview = Preview.Builder().build().also {
                it.setSurfaceProvider(previewView.surfaceProvider)
            }

            val cameraSelector = CameraSelector.Builder()
                .requireLensFacing(CameraSelector.LENS_FACING_BACK)
                .build()

            cameraProvider.unbindAll()
            cameraProvider.bindToLifecycle(
                lifecycleOwner,
                cameraSelector,
                preview
            )
        }, executor)
        previewView
    },
    modifier = Modifier.fillMaxSize(),
)
```

Set up the Analyzer

```
@androidx.camera.core.ExperimentalGetImage
class ImageAnalyzer(
    private val imageLabeler: ImageLabeler,
    private val onImageAnalyzerOutputs: (List<ImageLabel>) -> Unit,
) : ImageAnalysis.Analyzer {
    override fun analyze(imageProxy: ImageProxy) {
        imageProxy.image?.let {
            val image = InputImage.fromMediaImage(it, imageProxy.imageInfo.rotationDegrees)

            imageLabeler.process(image)
                .addOnSuccessListener { labels ->
                    onImageAnalyzerOutputs(labels)
                    imageProxy.close()
                }
        }
    }
}
```

```
@Composable
fun CameraPreview(
    onImageAnalyzerOutputs: (List<Category>) -> Unit
) {
    val lifecycleOwner = LocalLifecycleOwner.current
    val context = LocalContext.current
    val cameraProviderFuture = remember { ProcessCameraProvider.getInstance(context) }

    AndroidView(
        factory = { ctx ->
            ...

            // create the analyzer
            val analyzer = ImageAnalyzer(
                customLabeler,
             ) { outputs -> onImageAnalyzerOutputs(outputs) }

            cameraProviderFuture.addListener({
                val cameraProvider = cameraProviderFuture.get()
                val preview = Preview.Builder().build().also { setSurfaceProvider() }
                val cameraSelector = CameraSelector.Builder()
                    .requireLensFacing(CameraSelector.LENS_FACING_BACK)
                    .build()

                val imageAnalyzer = ImageAnalysis.Builder()
                    .setTargetResolution(Size(224, 224))
                    .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
                    .build()
```

```
AndroidView(
    factory = { ctx ->
        ...
        cameraProviderFuture.addListener({
            val cameraProvider =...
            val preview =...
            val cameraSelector =...

            // create ImageAnalysis object using the analyzer
            val imageAnalyzer = ImageAnalysis.Builder()
                    .setTargetResolution(Size(224, 224))
                    .setBackpressureStrategy(STRATEGY_KEEP_ONLY_LATEST)
                    .build()
                    .apply { setAnalyzer(executor, analyzer) }

            cameraProvider.unbindAll()

            // bind the image analyzer
            cameraProvider.bindToLifecycle(
                    lifecycleOwner,
                    cameraSelector,
                    preview,
                    imageAnalyzer
                )
        }, executor)
        previewView
```

If you’re stuck:

If stuck with the camera permissions - code is available here
If stuck with the analyzer - code is available here
If stuck with the camera permissions - code is available here

This code is completed in
6-main-activity-part-4

Once all that is implemented, we have a really cool app!

Let’s try it out!

Part 5 - Explore

Where to go from here

~~Maybe make a custom data model to label plants~~

~~Maybe use the camera so that I don’t have to load an image~~

Customize based on your interests (choose your own adventure)

Choose your own Adventure

If you liked building the UI:

Improve the UI of our app. Update the UI so that you can switch between Live camera mode and selecting an image, can pick which model to use (default or custom). Use colors and sizes to show likelihood of label based on confidence. This can all be done using state and Foundational & Material components and a little bit of logic.

If you want to dive deeper in Android and Jetpack Libraries, save these preferences to Preferences DataStore so they persist between sessions. Here, you'll need to use suspend functions and flow.

Choose your own Adventure

If you want to productionize your project:

Instead of shipping the model with our app (meaning new releases for model updates), we can host our model in firebase and download dynamically across our apps.

Create a new firebase project, add your android app, upload your model, and use the remote firebase config to pull it into your project. Follow the links below to get started.

Links: Label images with a custom model on Android, Add Firebase to your Android project, Firebase Machine Learning

Choose your own Adventure

If you want to know more about ML Kit:

Experiment more with our custom model.

Build on our custom model and add more flowers to it. Download the data set and add new labels by creating new folders. Experiment with the minimum number of images to add to detect the new label reliably.

Modify the parameters to see how high you can get the model's accuracy.

Go to TensorFlow Hub and find a model to use in your app.

Choose your own Adventure

If you want to know more about ML Kit:

Follow the MediaPipe Object Detection Model Customization Guide and build a custom object detection app.

Links: Media Pipe Object Detection Model Customization Guide and ML Kit Object Detection docs

Choose your own Adventure

If you want to know more about ML Kit:

All Vision APIs are set up in a really similar way, but return different data from the process function.

Browse the any of the other vision APIs, choose a new one that’s interesting to you, add it to your project, and explore how to use the data in a meaningful way.

Links: ML Kit docs to get started

Wrapping up

Where to go from here

~~Maybe make a custom data model to label plants~~

~~Maybe use the camera so that I don't have to load an image~~

Customize based on your interests (choose your own adventure)

Convince my dad to use an Android Phone so he can use it):

With great power comes great responsibility

Thank you!

Where can you find these materials?

https://github.com/sierraobryan/

Where can you find me?

@_sierraOBryan

https://sierraobryan.dev/

SierraOBryan

SIERRA OBRYAN

Questions?

A very happy client