Getting Started with ML Kit

SIERRA OBRYAN (SHE/HER)

@_sierraOBryan

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

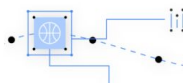Easy to Use

Okay.. But what can it do?

Vision APIs

Face Detection

Barcode Scanning

Selfie Segmentation

Object Tracking

Pose Detection

Digital Ink Recognition

Text Recognition

Image Labeling

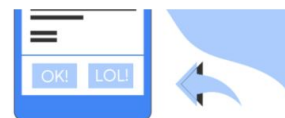## Natural Language APIs

Language ID

Entity Extraction

Smart Reply

OK! LOL!

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.
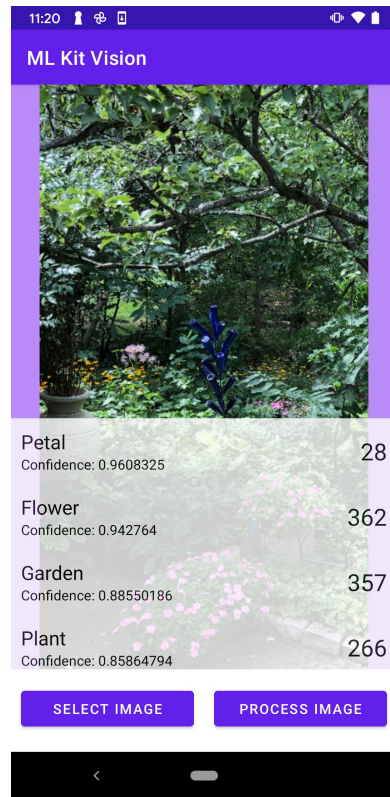
He really likes gardening.

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

He finally knows what I do for work.

He ask if I could write an app to help.

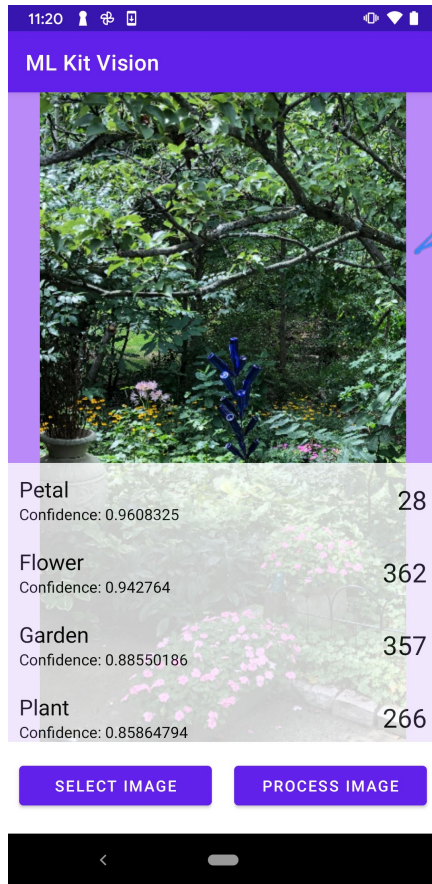# With ML Kit the answer is YES

Let's get our app started!

What are we building?

We're going to build a simple single Activity app

ConstraintLayout

ImageView

RecyclerView

Some Buttons

How do we use it?
We need to add the Image Labeling dependency to our gradle

```
dependencies {
  // ...
  // Use this dependency to bundle the model with your app
  implementation 'com.google.mlkit:image-labeling:17.0.2'
}
```

I'm going to bundle it with the app

```
dependencies {
  // ...
  // Use this dependency to use dynamically downloaded model in Google Play Service
  implementation 'com.google.android.gms:play-services-mlkit-image-labeling:16.0.2'
}
```

Clipper	Bonfire	Tuxedo	Beach
Nail	Comics	Mouth	Rainbow
Cola	Himalayan	Desert	Branch
Cutlery	Iceberg	Dinosaur	Moustache
Menu	Bento	Mufti	Garden
Sari	Sink	Fire	Gown
Plush	Toy	Bedroom	Field
Pocket	Statue	Goggles	Dog
Neon	Cheeseburger	Dragon	Superhero
Icicle	Tractor	Couch	Flower
Pasteles	Sled	Sledding	Placemat
Chain	Aquarium	Cap	Subwoofer
Dance	Circus	Whiteboard	Cathedral
Dune	Sitting	Hat	Building
Santa claus	Beard	Gelato	Airplane
Thanksgiving	Bridge	Cavalier	Fur
Tuxedo	Tights	Beanie	Bull
Mouth	Bird	Jersey	Bench
Desert	Rafting	Scarf	Temple
Dinosaur	Park	Vacation	Butterfly
Mufti	Factory	Pitch	Model
Fire	Graduation	Blackboard	

We're just going to use the Base Model

Step One
Prepare the Input Image

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

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

```kotlin
private fun startChooseImageIntentForResult () {
    val intent = Intent()
    intent.type = "image/*"
    intent.action = Intent.ACTION_GET_CONTENT
    startActivityForResult(
        Intent.createChooser(intent, "Select
Picture"),
        REQUEST_CHOOSE_IMAGE
    )
}

override fun onActivityResult(
    requestCode: Int,
    resultCode: Int,
    data: Intent?
) {
    onSelectImageResult(data?. data != null)
    if (requestCode == REQUEST_CHOOSE_IMAGE &&
        resultCode == Activity.RESULT_OK
    ) {
        val imageUri = data!!.data
        setPreview(imageUri)
    } else {
        super.onActivityResult(
            requestCode, resultCode, data
        )
    }
}
```

Step One
Prepare the Input Image

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

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

```kotlin
private fun setPreview(imageUri: Uri?) {
    try {
        if (imageUri == null) return

        val preview = findViewById<ImageView>(R.id.preview)

        val imageBitmap = getBitmapFromUri(imageUri) ?: return

        this.imageBitmap = imageBitmap

        preview.setImageBitmap(imageBitmap)

    } catch (e: IOException) {
        Toast.makeText(this,
            getString(R.string.something_went_wrong),
            Toast.LENGTH_SHORT
        ).show()
    }
}

@Throws(IOException::class)
private fun getBitmapFromUri(uri: Uri): Bitmap? {
    val parcelFileDescriptor =
        contentResolver.openFileDescriptor(uri, "r")
    val fileDescriptor = parcelFileDescriptor?.fileDescriptor
    val image = BitmapFactory
        .decodeFileDescriptor(fileDescriptor)
    parcelFileDescriptor?.close()
    return image
}
```

Process the image

Now that we have our bitmap, we can convert that to an imageInput.

Then we can create our labeler. In this case, we're just using the default options.

```kotlin
if (imageBitmap != null) {
    val imageInput = InputImage.fromBitmap(imageBitmap!!, 0)

    val labeler = ImageLabeling.getClient(ImageLabelerOptions.DEFAULT_OPTIONS)

    labeler.process(imageInput).addOnSuccessListener { labels ->

        // do something with our labels

    }.addOnFailureListener {
        Toast.makeText(this, getString(R.string.nothing_found), Toast.LENGTH_SHORT).show()
    }
}
```

Finally, we can can process our imageInput with our labeler

We add an onSuccessListener for when it works YAY!

And a onFailureListener for when it doesn't):

Send our labels to our view

```kotlin
if (imageBitmap != null) {
    val imageInput = InputImage.fromBitmap(imageBitmap!!, 0)

    val labeler = ImageLabeling.getClient(ImageLabelerOptions.DEFAULT_OPTIONS)

    labeler.process(imageInput).addOnSuccessListener { labels ->

        val recyclerView = findViewById<RecyclerView>(R.id.labels)
        recyclerView.layoutManager = LinearLayoutManager(this)
        recyclerView.adapter = LabelAdapter(labels)
        recyclerView.visibility = View.VISIBLE

    }.addOnFailureListener {
        Toast.makeText(this, getString(R.string.nothing_found), Toast.LENGTH_SHORT).show()
    }
}
```

When we successfully process our image, we get back a list of labels

For our simple app, we're going to display our list of labels in a recyclerView so we pass them into an adapter

Display the Labels

Each ImageLabel
(in our list of ImageLabels)
has a

Text (String)
Confidence (Float)
Index (Integer)

```kotlin
fun bind(imageLabel: ImageLabel) {
    label.text = imageLabel.text
    confidence.text = String.format(
        itemView.resources.getString(R.string.confidence_format),
        imageLabel.confidence.toString()
    )
    index.text = imageLabel.index.toString()
}
```

In our simple example, we bind those to our row view and we're good to go!

And with that we have an app!

Let's try it out!
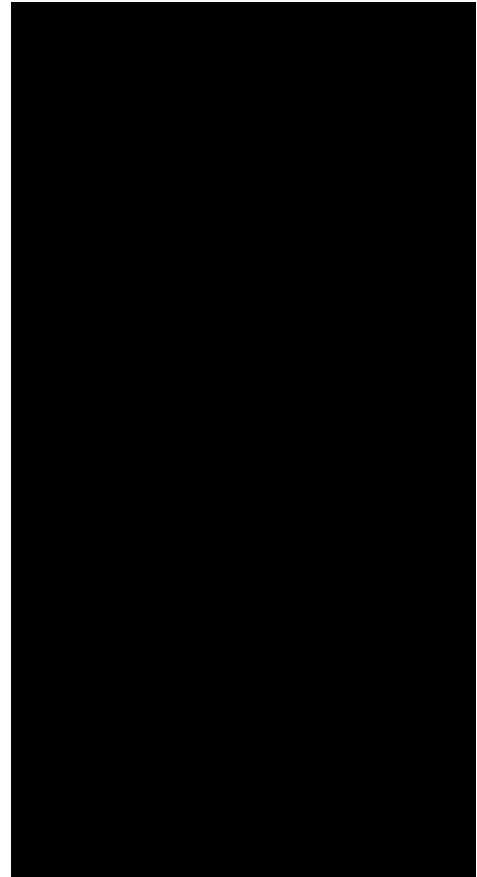
Where to go from here

Maybe make a custom data model to label plants

Let's do it!

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
- AutoML Vision Edge
- TensorFlow Lite 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
- AutoML Vision Edge
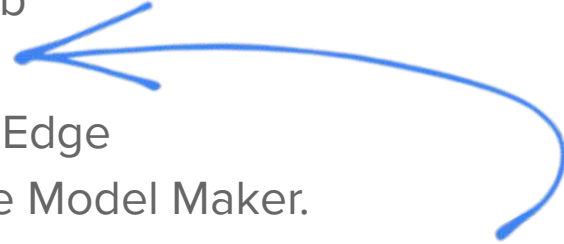- TensorFlow Lite Model Maker.

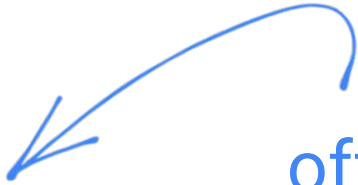offers a wide range of pre-trained image classification models

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
- AutoML Vision Edge
- TensorFlow Lite 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
- AutoML Vision Edge
- TensorFlow Lite Model Maker.

offered through Google Cloud AI

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 Vision Edge
- TensorFlow Lite Model Maker.

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

We're going to use this model maker

Creating a Custom Model

▾ Prerequisites

To run this example, we first need to install several required packages, including Model Maker package that in GitHub repo.

```
[ ]  !pip install tflite-model-maker
```

Import the required packages.

```
[ ]  import os

     import numpy as np

     import tensorflow as tf
     assert tf.__version__.startswith('2')

     from tflite_model_maker import configs
     from tflite_model_maker import ExportFormat
     from tflite_model_maker import image_classifier
     from tflite_model_maker import ImageClassifierDataLoader
     from tflite_model_maker import model_spec

     import matplotlib.pyplot as plt
```
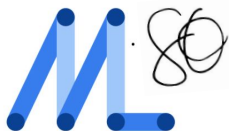
Creating a Custom Model

```python
image_path = tf.keras.utils.get_file(
    'Flower_photos.tgz',
     'https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz',
    extract=True)
image_path = os.path.join(os.path.dirname(image_path), 'flower_photos')
```

Creating a Custom Model

First we load the data

```
data = ImageClassifierDataLoader.from_folder(image_path)
   INFO:tensorflow:Load image with size: 3670, num_label: 5, labels: daisy,
   dandelion, roses, sunflowers, tulips.
train_data, test_data = data.split(0.9)


model = image_classifier.create(train_data)
```

Then we split our data into 90% training and 10% test

Create a custom image classifier model from our training data

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


model.export(export_dir='.')
```

Use our test data to test evaluate the accuracy of model

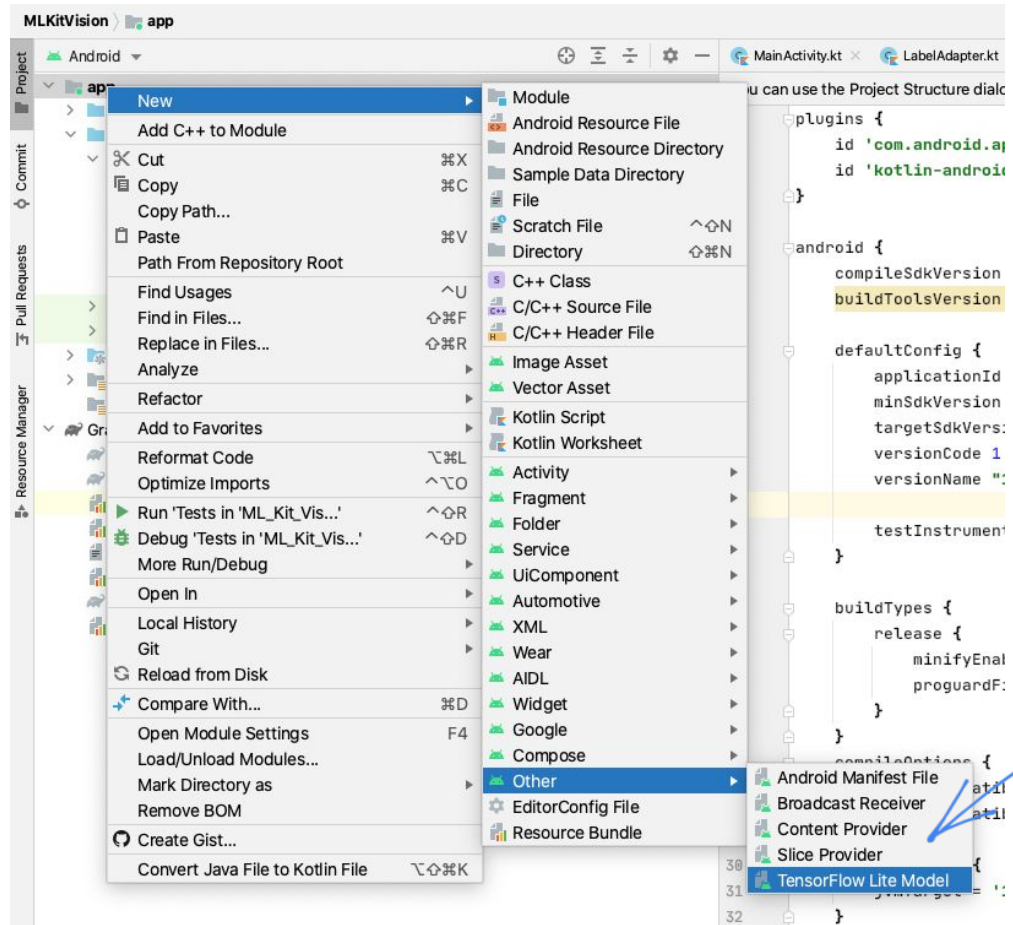We're done! Export our model!

Creating a Custom Model

```
model = image_classifier.create(train_data)
```
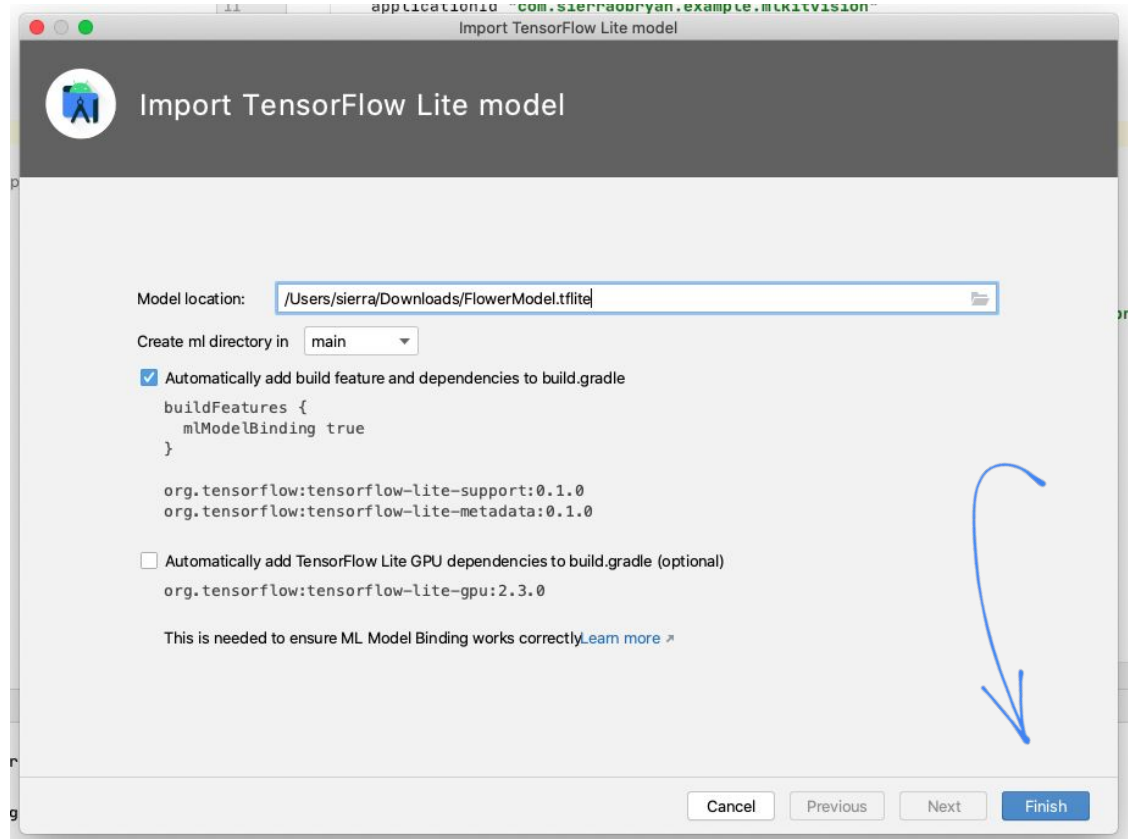
Adding the model to your project

All the way
down here

Adding the model to your project

Adding the model to your project

```
kotlinOptions {                                          30    30    kotlinOptions {
    jvmTarget = '1.8'                                    31    31        jvmTarget = '1.8'
}                                                        32    32    }
}                                                     » 33    33    buildFeatures {
                                                         34    34        mlModelBinding true
                                                         35    35    }
dependencies {                                           36    36  }

                                                         37    37
    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"   38   38  dependencies {
    implementation 'androidx.core:core-ktx:1.3.2'        39    39
    implementation 'androidx.appcompat:appcompat:1.2.0'  40    40        implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
    implementation 'com.google.android.material:material:1.3.0'          41   41        implementation 'androidx.core:core-ktx:1.3.2'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'               implementation 'androidx.appcompat:appcompat:1.2.0'
    testImplementation 'junit:junit:4.+'              » 42    42        implementation 'com.google.android.material:material:1.3.0'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'            43   43        implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'  44  44
                                                         45    45        implementation 'org.tensorflow:tensorflow-lite-support:0.1.0'
    implementation 'com.google.mlkit:image-labeling:17.0.2'             46   46        implementation 'org.tensorflow:tensorflow-lite-metadata:0.1.0'
}                                                        47    47        testImplementation 'junit:junit:4.+'
                                                         48          androidTestImplementation 'androidx.test.ext:junit:1.1.2'
                                                         49          androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
                                                         50
                                                         51          implementation 'com.google.mlkit:image-labeling:17.0.2'
                                                         52  }
```

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
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!

This time we'll convert our bitmap to a TensorImage

```kotlin
val tfImage = TensorImage.fromBitmap(bitmap)
```

We'll create a new instance of our Model

```kotlin
val flowerModel = FlowerModel.newInstance(this)
```

And now we'll process

```kotlin
val outputs = flowerModel.process(tfImage)
    .probabilityAsCategoryList.apply {
        sortByDescending { it.score }
    }

if (outputs.isNotEmpty()) {
    val recyclerView = findViewById<RecyclerView>(R.id. labels)
    recyclerView. layoutManager = LinearLayoutManager(this)
    recyclerView. adapter = TFImageAdapter(outputs)
    recyclerView. visibility = View.VISIBLE
}
```

Let's use our new model!

```kotlin
fun bind(category: Category) {
    label.text = category.label
    confidence.text = String.format(
        itemView.resources.getString(R.string.confidence_format),
        category.score * 100
    )
}
```

This time we're going to get a list of Categories passed into our adapter

Each Category has a label and a confidence

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

Let's try it out!
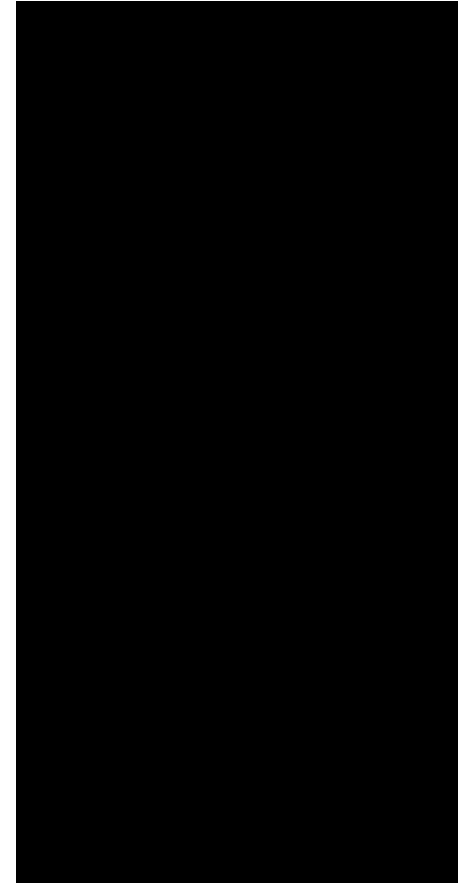
And on our original backyard image

Let's try it out!
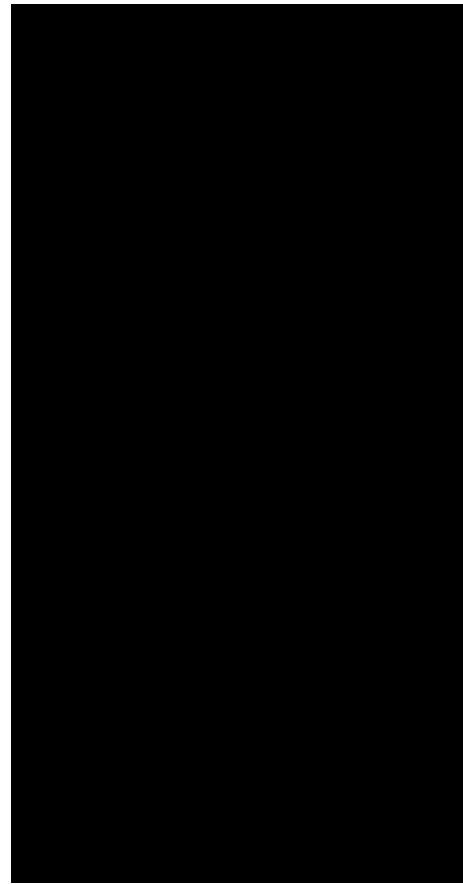
YAY! We can label five flowers

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
        |_____ ...
```
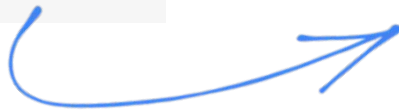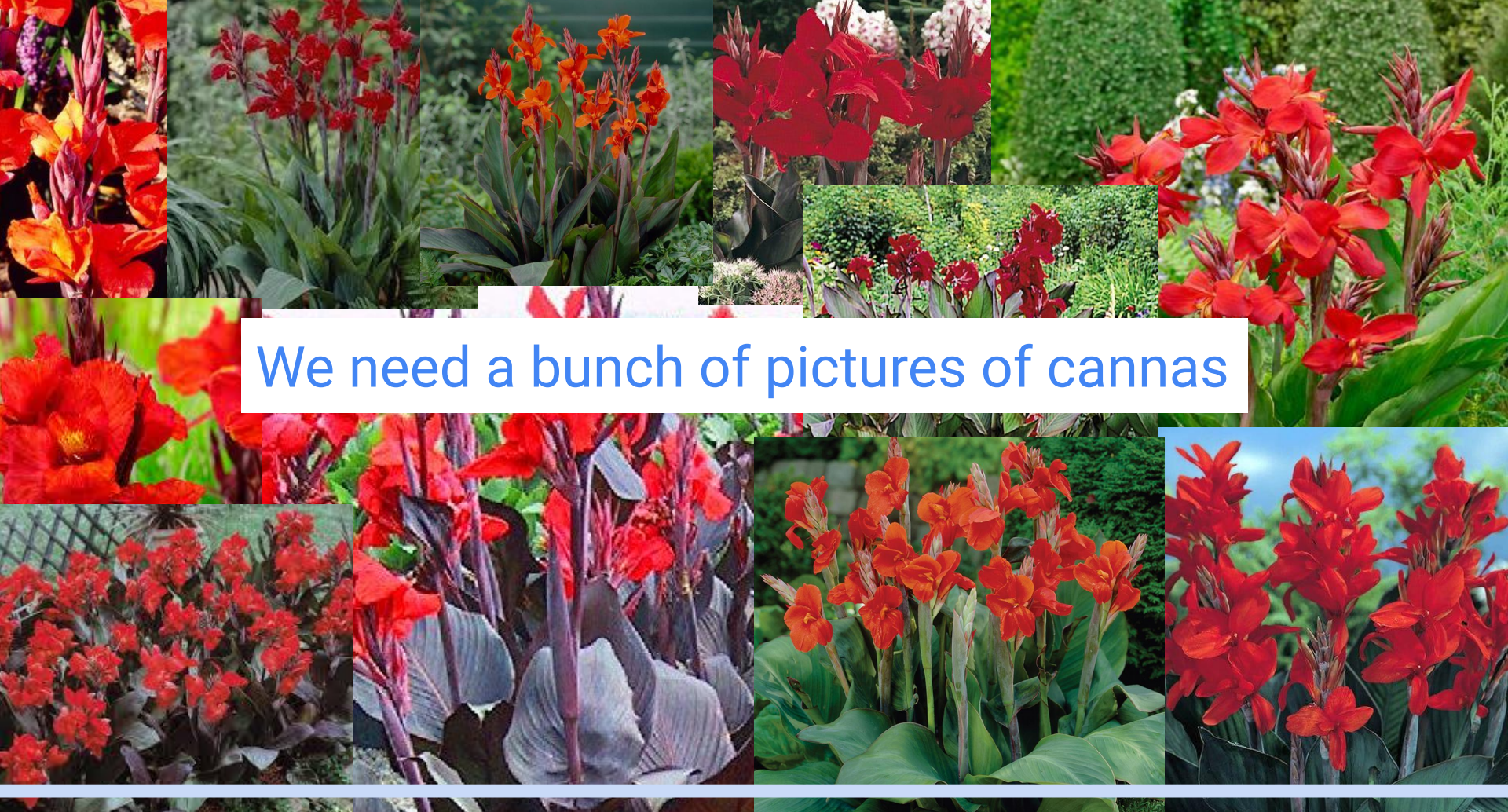
We need a bunch of pictures of cannas

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')
```

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

Let's try it out!
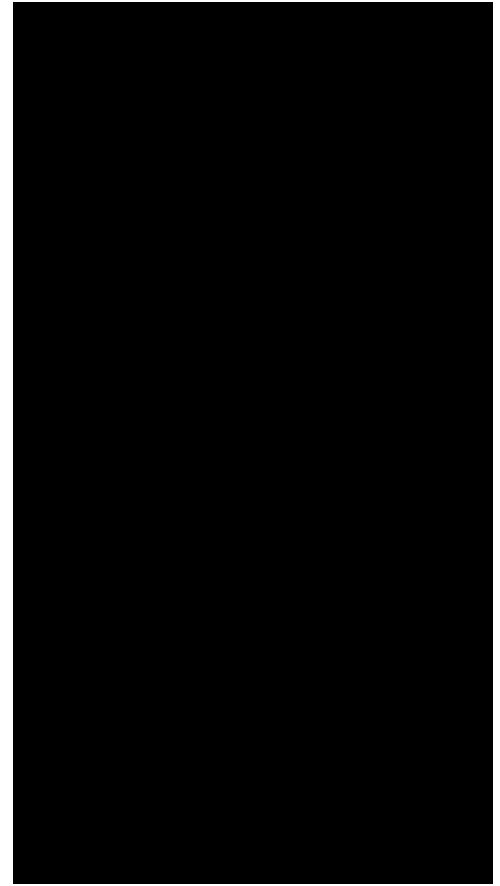
Where to go from here

~~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!

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 activity 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 RecyclerView (and display in the PreviewView)
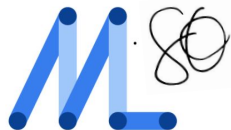
There's a lot happening in this sentence - this is completed on branch ml-kit-camera

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

Let's try it out!
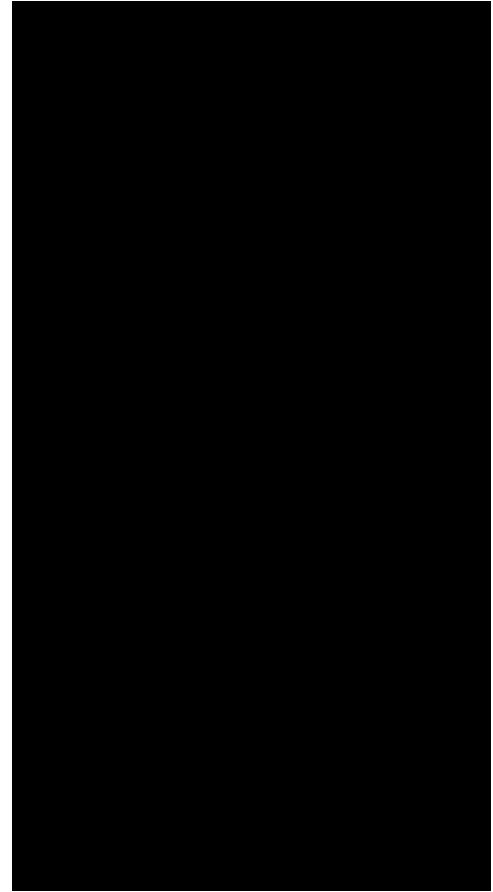
Where to go from here

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

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

Maybe overlay the image with the labels

Rewrite in Jetpack Compose!

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 this code?

https://github.com/sierraobryan/examples/tree/main/MLKitVision

Where can you find me?

🐦 @_sierraOBryan

https://sierraobryan.com/