Lab session 6: Integrating Machine Learning Model in a Flutter application

Task-1 TensorFlow Lite Tutorial for Flutter: Image Classification

In this task, we will learn how to use TensorFlow Lite in Flutter. This implies training the machine learning model with **Teachable Machine** and integrate the result into a Flutter mobile app.

We will develop an application called Plant Recognizer that uses machine learning to recognize plants simply by looking at photos of them. We will accomplish this by using the Teachable Machine platform, TensorFlow Lite, and a Flutter package named tflite_flutter.

TensorFlow is a popular machine-learning library for developers who want to build learning models for their apps. TensorFlow Lite is a mobile version of TensorFlow for deploying models on mobile devices. And Teachable Machine is a beginner-friendly platform for training machine learning models.

Getting Started

We will use a starter project provided by the tutorial as a base. The project already allows users to pick images or drag and drop it directly to the app but the app doesn't recognize images.

We will use TensorFlow Lite to solve that in the next sections.

Building a Model with Teachable Machine

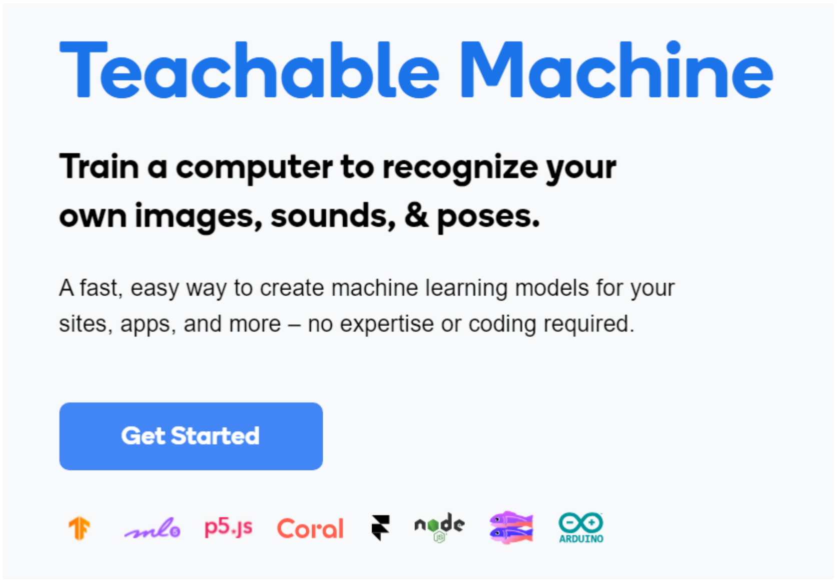
The starter project already contains a trained model model_unquant.tflite and classification labels in the labels.txt file but we will start from the training process to understand how it is implemented.

Preparing the Dataset- Training the Model

Training is the process by which the computer learns data and derives rules.

We will learn how to train the model using Teachable Machine.

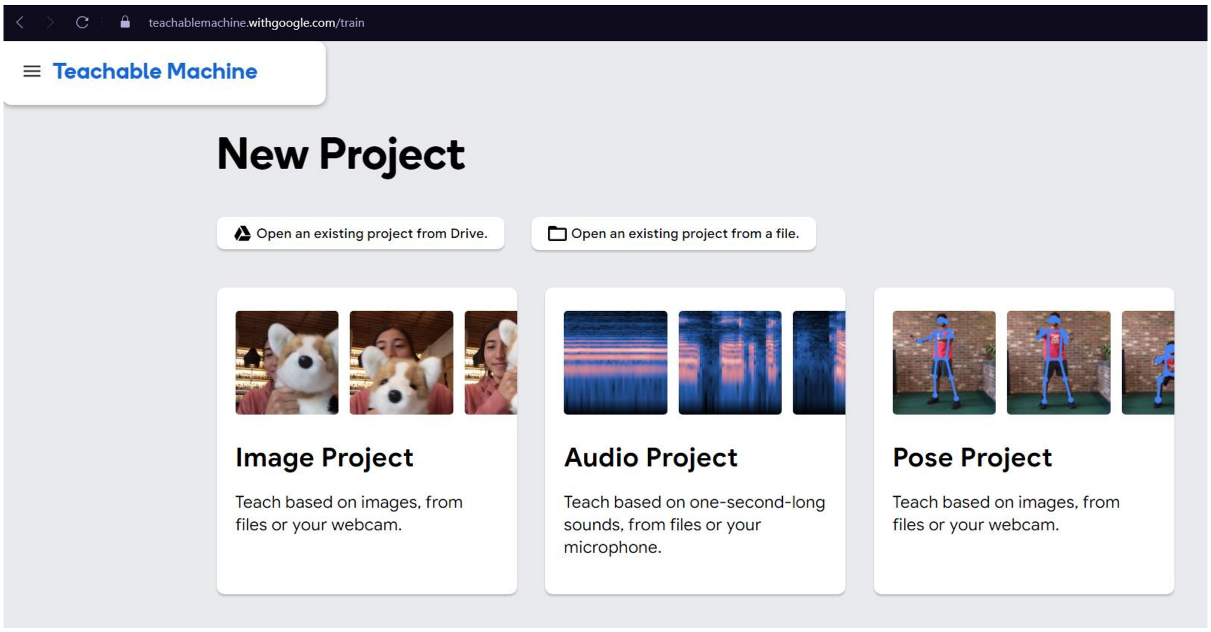
1. First, go to https://teachablemachine.withgoogle.com and click Get Started to open the training tool:


Teachable Machine

Train a computer to recognize your own images, sounds, & poses.

A fast, easy way to create machine learning models for your sites, apps, and more – no expertise or coding required.

2. Then select Image Project:


teachablemachine.withgoogle.com/train
Teachable Machine
Open an existing project from Drive.
Open an existing project from a file.

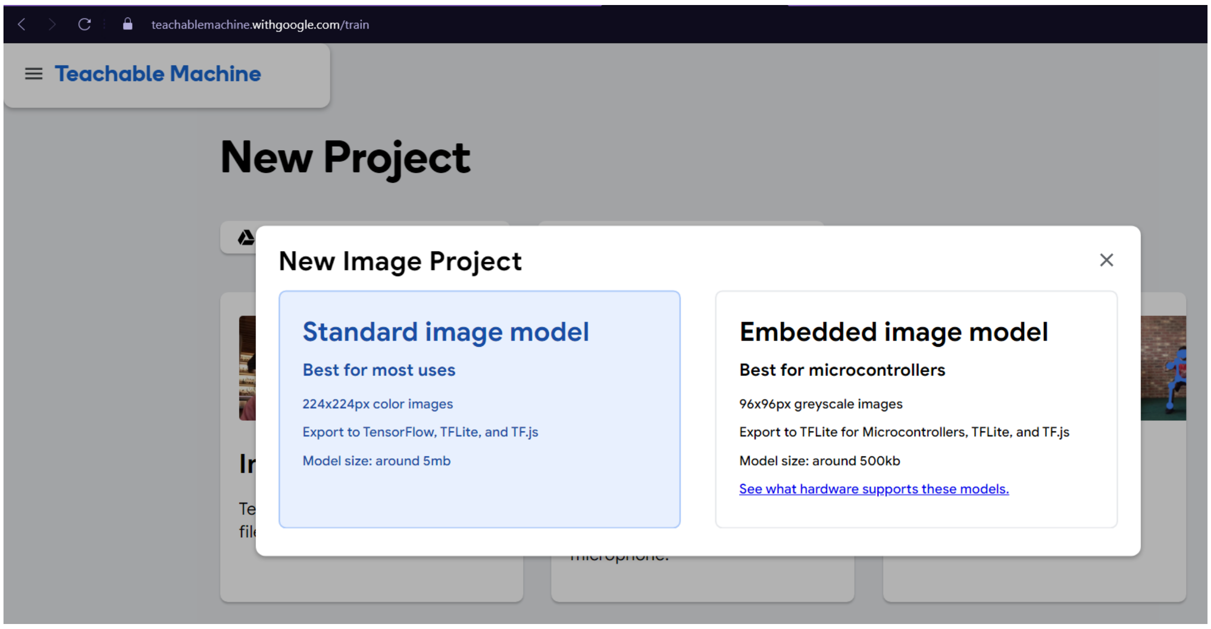
New Project

Image Project
Teach based on images, from files or your webcam.

Audio Project
Teach based on one-second-long sounds, from files or your microphone.

Pose Project
Teach based on images, from files or your webcam.

3. Choose Standard Image Model, because we are not training a model to run on a microcontroller:

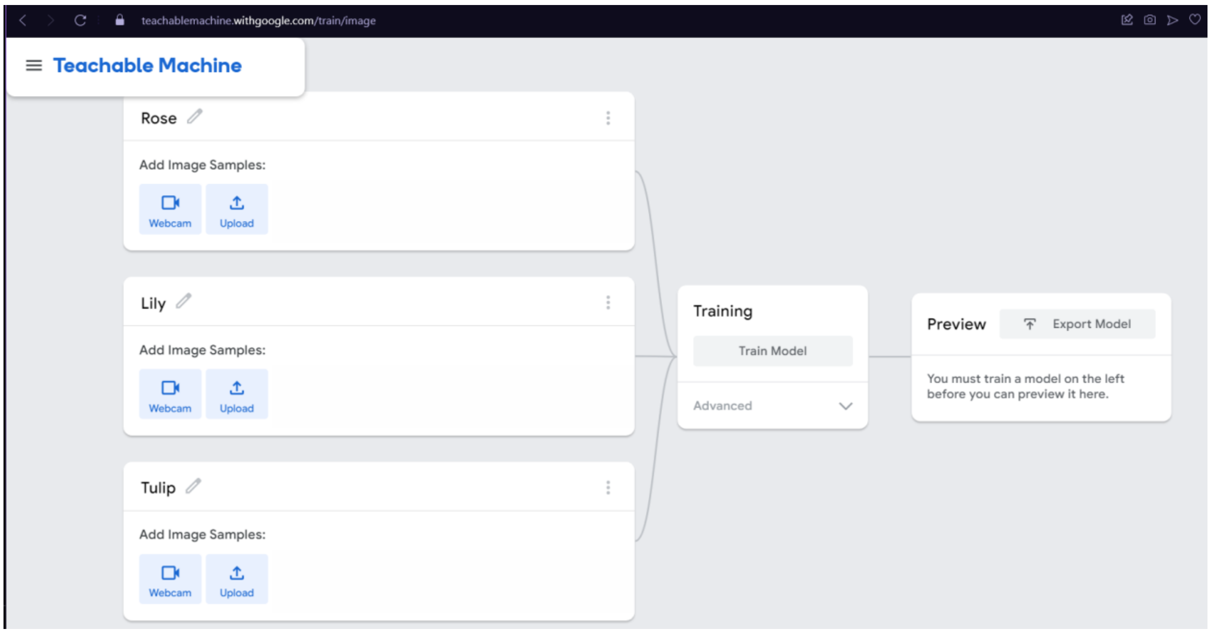


New Image Project

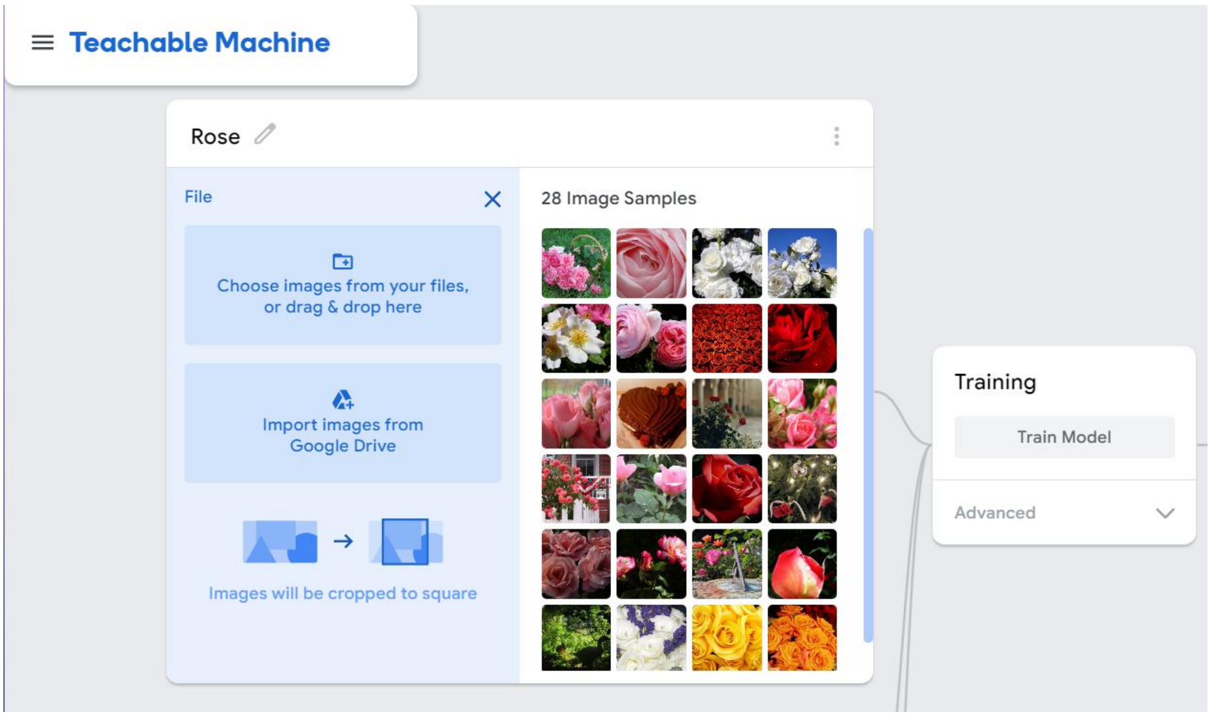
Standard image model
Best for most uses
224x224px color images
Export to TensorFlow, TFLite, and TF.js
Model size: around 5mb

Embedded image model
Best for microcontrollers
96x96px greyscale images
Export to TFLite for Microcontrollers, TFLite, and TF.js
Model size: around 500kb
See what hardware supports these models.

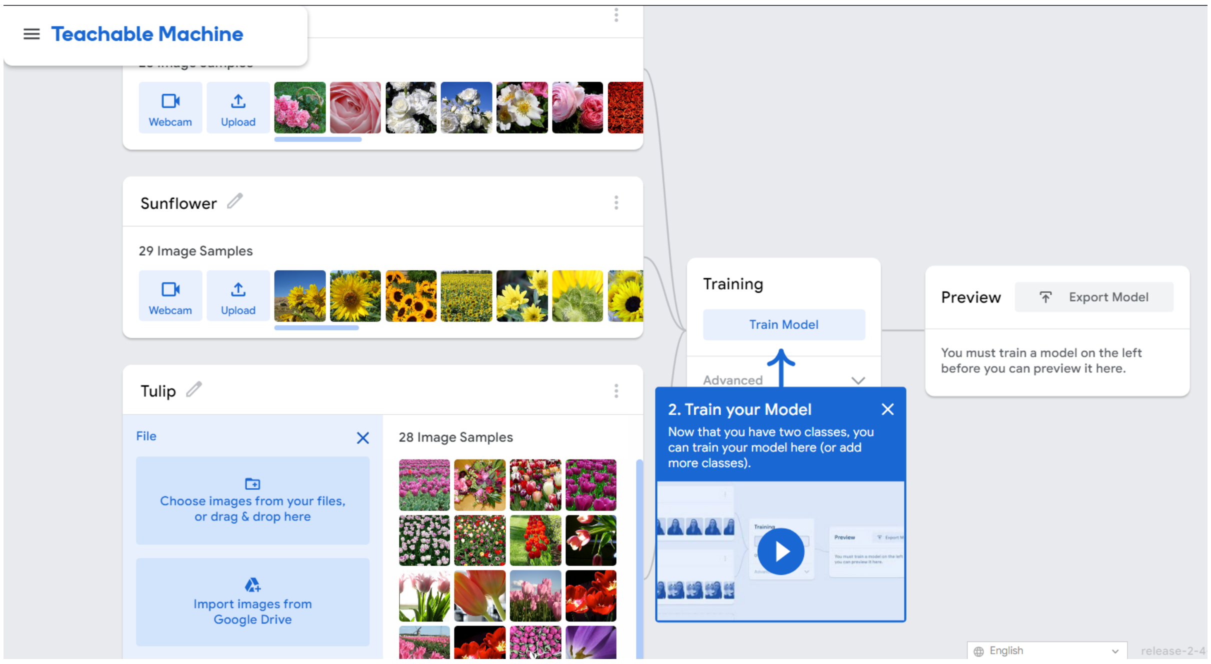
4. Once in the training tool, add the classes and edit the labels of each class, as shown below:



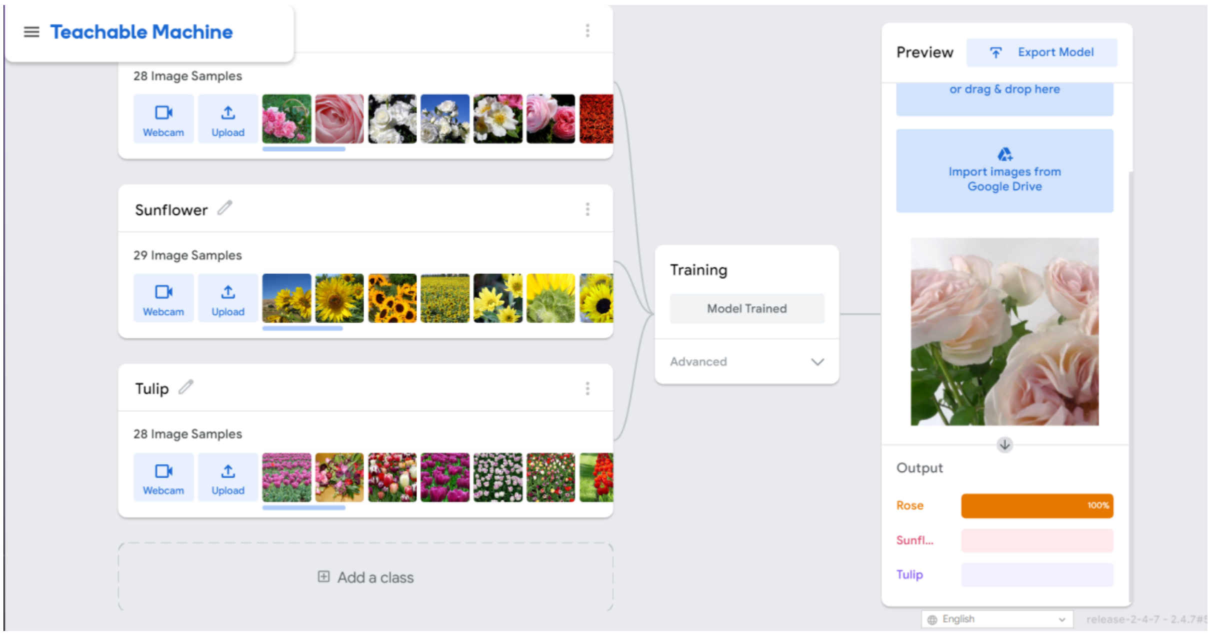
5. Next, add the training samples by clicking Upload under each class. Then, drag the folder of the appropriate plant type from the samples folder to the Choose images from your files … panel.


Teachable Machine
Rose
File
Choose images from your files, or drag & drop here
Import images from Google Drive
Images will be cropped to square
28 Image Samples
Training
Train Model
Advanced

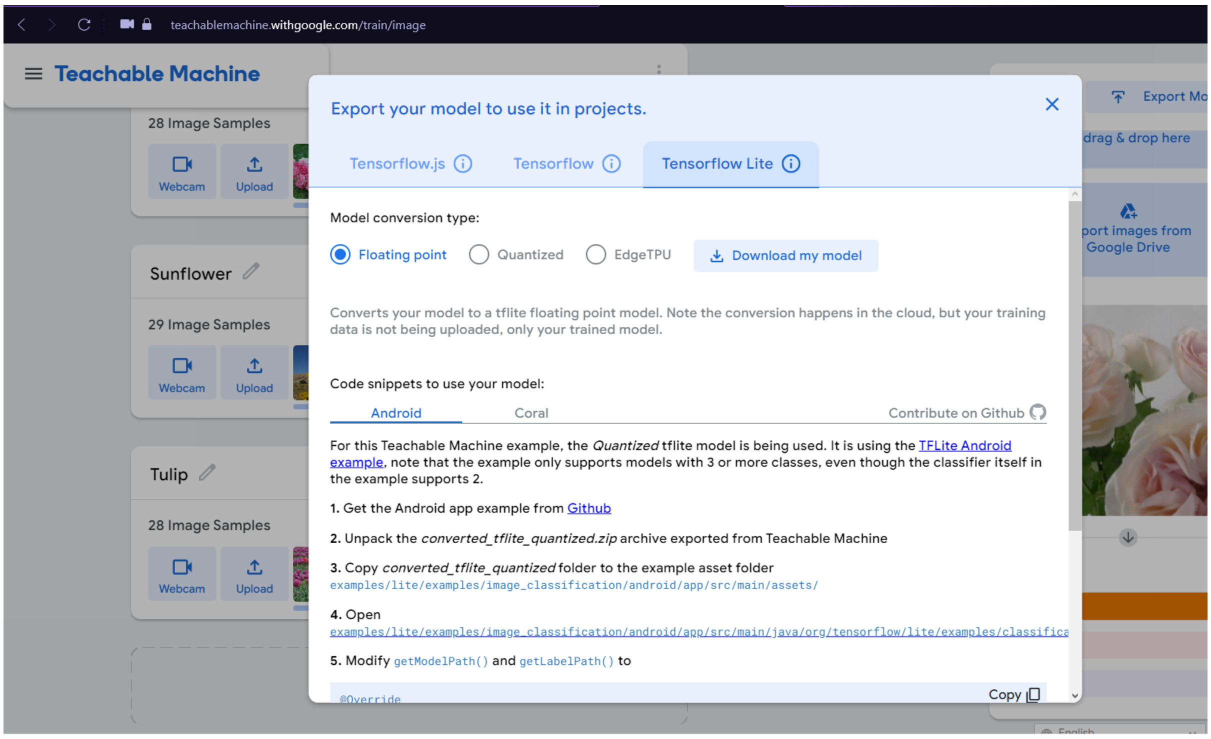
6. After you've added all the training samples, click Train Model to train the model:


Teachable Machine
Webcam
Upload
Sunflower
29 Image Samples
Tulip
File
Choose images from your files, or drag & drop here
Import images from Google Drive
28 Image Samples
Training
Train Model
Advanced
2. Train your Model
Now that you have two classes, you can train your model here (or add more classes).
Preview
Export Model
You must train a model on the left before you can preview it here.
English

7. After the training completes, test the model with other plant images. Use the images in the samples-test folder, like so:



8. Finally, export the model by clicking Export Model on the Preview panel. In the dialog, choose TensorFlow Lite. That’s because the target platform is mobile.
9. Next, select Floating point conversion type for the best predictive performance. Then, click Download my model to convert and download the model.



Export your model to use it in projects.

Model conversion type:

[✓ Floating point] Quantized EdgeTPU — Download my model

Converts your model to a tflite floating point model. Note the conversion happens in the cloud, but your training data is not being uploaded, only your trained model.

Code snippets to use your model:

Android | Coral | Contribute on Github

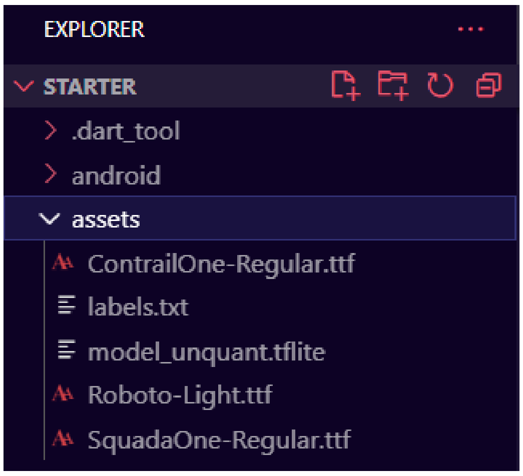
For this Teachable Machine example, the *Quantized* tflite model is being used. It is using the TFLite Android example, note that the example only supports models with 3 or more classes, even though the classifier itself in the example supports 2.

1. Get the Android app example from Github
2. Unpack the *converted_tflite_quantized.zip* archive exported from Teachable Machine
3. Copy *converted_tflite_quantized* folder to the example asset folder `examples/lite/examples/image_classification/android/app/src/main/assets/`
4. Open `examples/lite/examples/image_classification/android/app/src/main/java/org/tensorflow/lite/examples/classifica`
5. Modify `getModelPath()` and `getLabelPath()` to

```
@Override
```

It may take several minutes to complete the model conversion process. Once it's done, the model file will automatically download to your system.

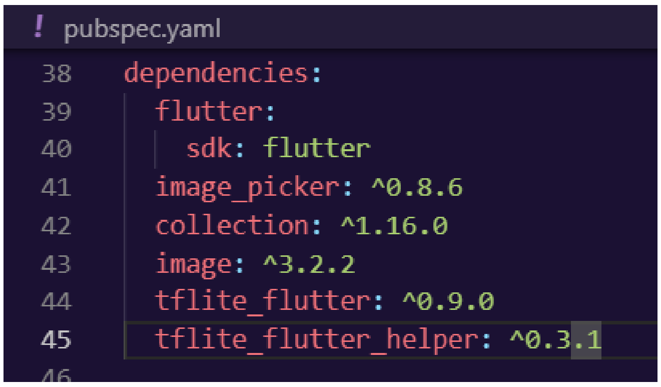
10. After you have the model file converted_tflite.zip in hand, decompress it and copy labels.txt and model_unquant.tflite to the ./assets folder in the starter project.



```
EXPLORER
STARTER
  > .dart_tool
  > android
  v assets
      ContrailOne-Regular.ttf
      labels.txt
      model_unquant.tflite
      Roboto-Light.ttf
      SquadaOne-Regular.ttf
```

Installing TensorFlow Lite in Flutter

To use TensorFlow in your Flutter app, you need to install the following packages:


pubspec.yaml

```yaml
dependencies:
  flutter:
    sdk: flutter
  image_picker: ^0.8.6
  collection: ^1.16.0
  image: ^3.2.2
  tflite_flutter: ^0.9.0
  tflite_flutter_helper: ^0.3.1
```

- tflite_flutter: allows you to access the native TensorFlow Lite library. When you invoke the methods of tflite_flutter, it calls the corresponding method of the native TensorFlow Lite SDK.
- tflite_flutter_helper: enables you to manipulate TensorFlow inputs and outputs. For example, it converts image data to tensor structure. It reduces the effort required to create pre- and post-processing logic for your model.

Creating an Image Classifier

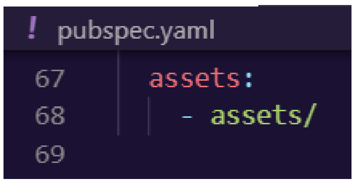
In machine learning, classification refers to predicting the class of an object out of a finite number of classes, given some input.

The starter project already implements the widgets and usage of the Classifier instance.

Importing the Model to Flutter

There are two pieces of data that we will load into the program: the machine learning model – model_unquant.tflite and the classification labels — labels.txt, which we got from the Teachable Machine platform.

To begin, make sure to include the assets folder in pubspec.yaml:

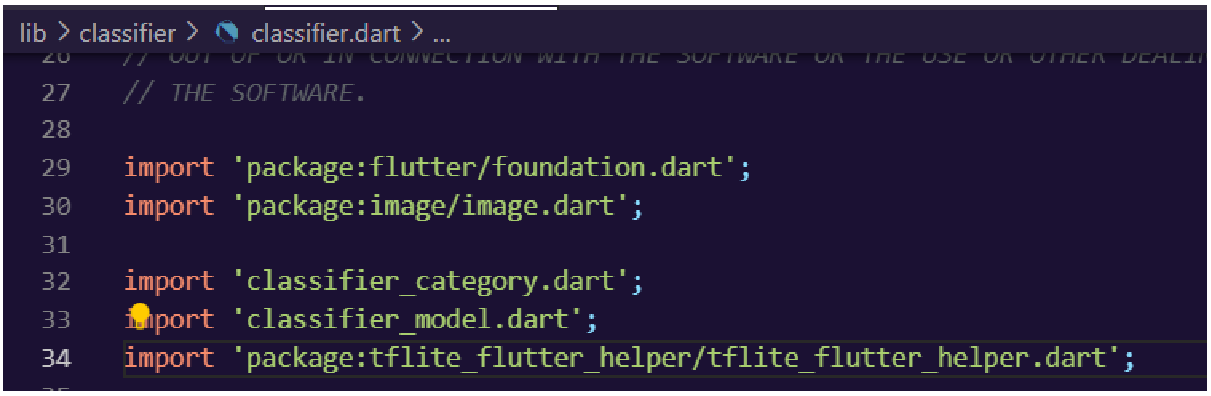

pubspec.yaml

```yaml
  assets:
    - assets/
```

The assets record is responsible for copying the resource files to the final application bundle.

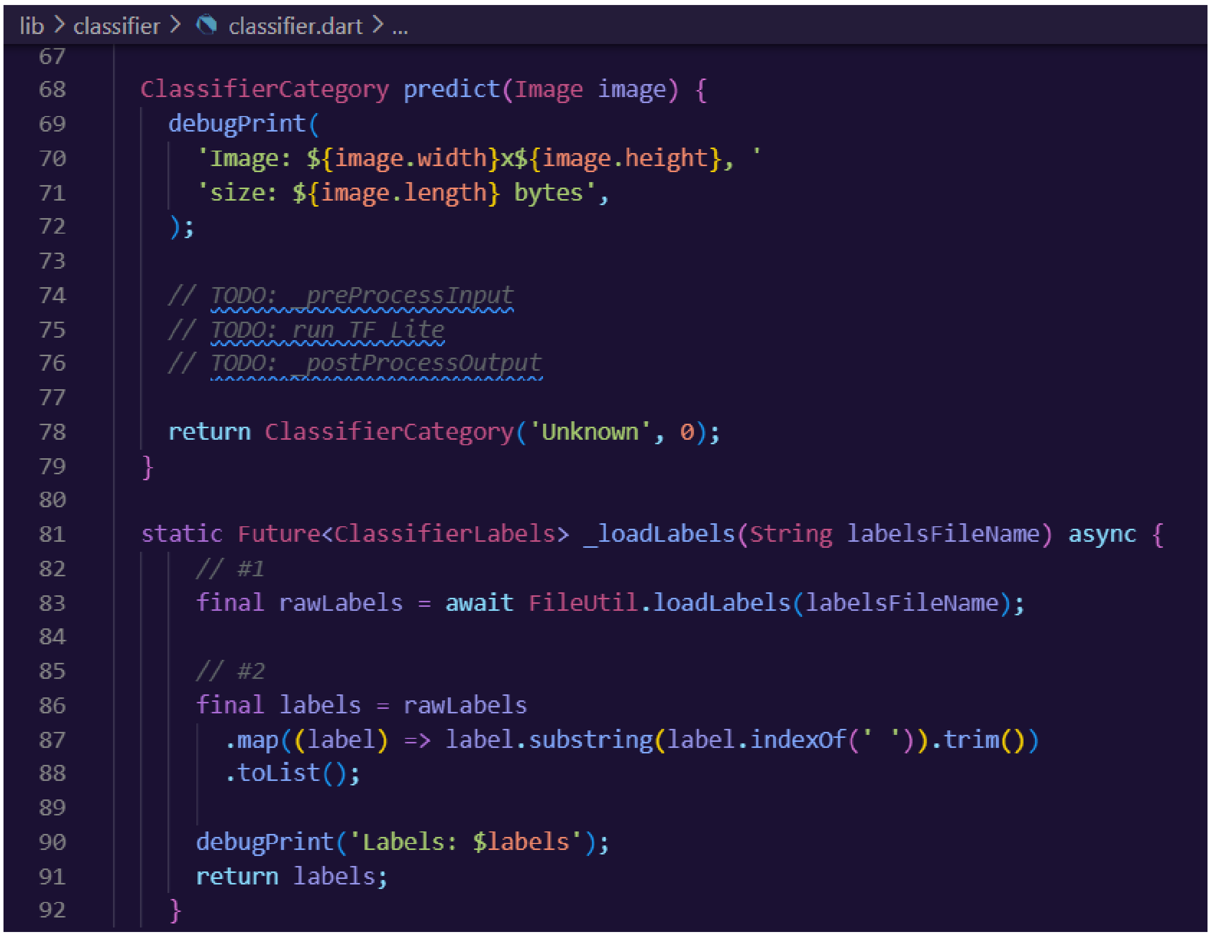
Loading Classification Labels

1. Open lib/classifier/classifier.dart and import tflite_flutter_helper:


lib > classifier > classifier.dart > ...

```dart
// THE SOFTWARE.

import 'package:flutter/foundation.dart';
import 'package:image/image.dart';

import 'classifier_category.dart';
import 'classifier_model.dart';
import 'package:tflite_flutter_helper/tflite_flutter_helper.dart';
```

2. Then add the following code after predict:

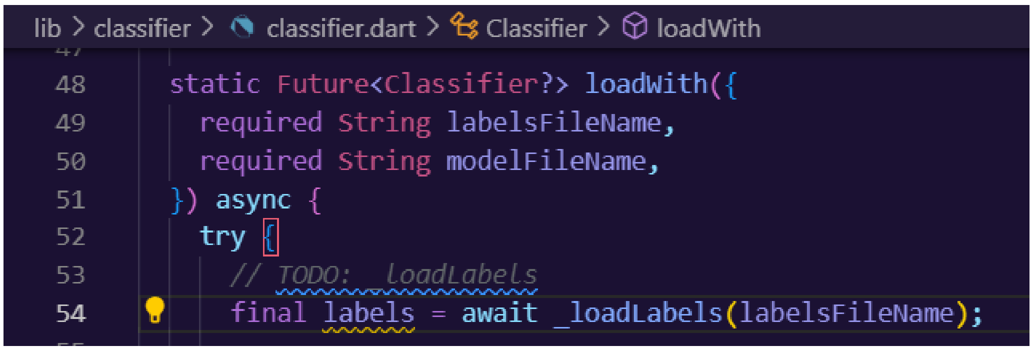

lib > classifier > classifier.dart > ...

```dart

  ClassifierCategory predict(Image image) {
    debugPrint(
      'Image: ${image.width}x${image.height}, '
      'size: ${image.length} bytes',
    );

    // TODO: _preProcessInput
    // TODO: run TF Lite
    // TODO: _postProcessOutput

    return ClassifierCategory('Unknown', 0);
  }

  static Future<ClassifierLabels> _loadLabels(String labelsFileName) async {
    // #1
    final rawLabels = await FileUtil.loadLabels(labelsFileName);

    // #2
    final labels = rawLabels
      .map((label) => label.substring(label.indexOf(' ')).trim())
      .toList();

    debugPrint('Labels: $labels');
    return labels;
  }
```

Here's what the above code does:

- Loads the labels using the file utility from **tflite_flutter_helper**.
- Removes the index number prefix from the labels you previously downloaded. For example, it changes **0 Rose** to **Rose**.

3. Next, replace // TODO: _loadLabels in loadWith by calling _loadLabels like so:

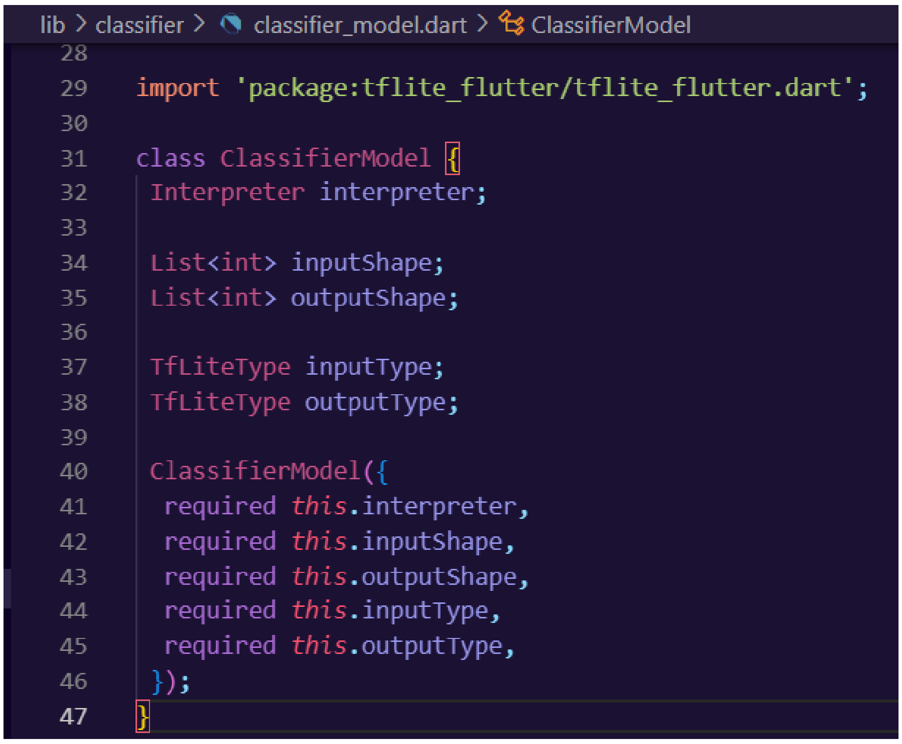

lib > classifier > classifier.dart > Classifier > loadWith

```dart
  static Future<Classifier?> loadWith({
    required String labelsFileName,
    required String modelFileName,
  }) async {
    try {
      // TODO: _loadLabels
      final labels = await _loadLabels(labelsFileName);
```

This code loads the label file.

Importing TensorFlow Lite Model

1. Go to lib/classifier/classifier_model.dart and replace the contents with the following code:

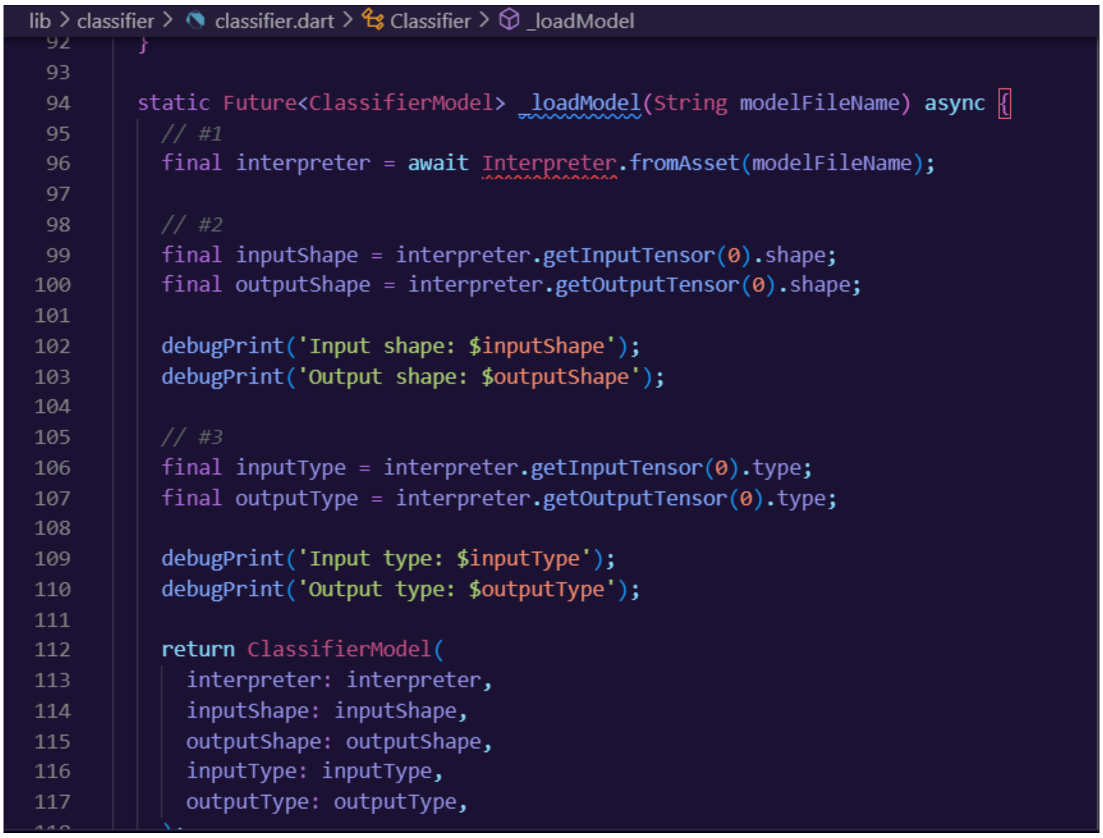


lib > classifier > classifier_model.dart > ClassifierModel

```dart
import 'package:tflite_flutter/tflite_flutter.dart';

class ClassifierModel {
  Interpreter interpreter;

  List<int> inputShape;
  List<int> outputShape;

  TfLiteType inputType;
  TfLiteType outputType;

  ClassifierModel({
    required this.interpreter,
    required this.inputShape,
    required this.outputShape,
    required this.inputType,
    required this.outputType,
  });
}
```

ClassifierModel stores all model-related data for your classifier. You'll use the interpreter to predict the results. inputShape and outputShape are shapes for the input and output data respectively while inputType and outputType are the data types of the input and output tensors.

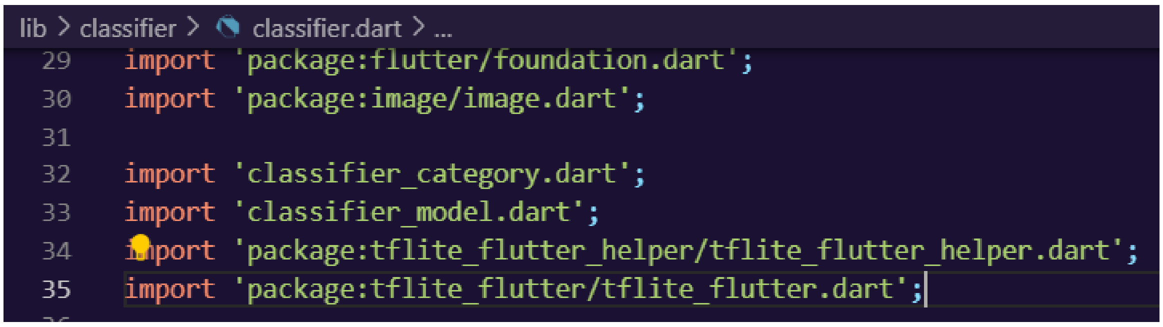
2. Now, import the model from the file. Go to lib/classifier/classifier.dart and add the following code after _loadLabels:



lib > classifier > classifier.dart > Classifier > _loadModel

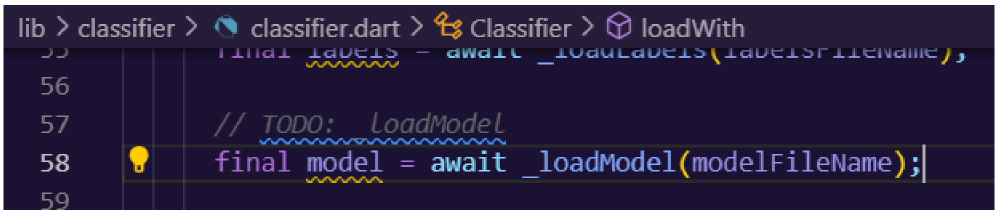
```dart
  }

  static Future<ClassifierModel> _loadModel(String modelFileName) async {
    // #1
    final interpreter = await Interpreter.fromAsset(modelFileName);

    // #2
    final inputShape = interpreter.getInputTensor(0).shape;
    final outputShape = interpreter.getOutputTensor(0).shape;

    debugPrint('Input shape: $inputShape');
    debugPrint('Output shape: $outputShape');

    // #3
    final inputType = interpreter.getInputTensor(0).type;
    final outputType = interpreter.getOutputTensor(0).type;

    debugPrint('Input type: $inputType');
    debugPrint('Output type: $outputType');

    return ClassifierModel(
      interpreter: interpreter,
      inputShape: inputShape,
      outputShape: outputShape,
      inputType: inputType,
      outputType: outputType,
```

Remember to add the import import 'package:tflite_flutter/tflite_flutter.dart'; at the top.



```dart
import 'package:flutter/foundation.dart';
import 'package:image/image.dart';

import 'classifier_category.dart';
import 'classifier_model.dart';
import 'package:tflite_flutter_helper/tflite_flutter_helper.dart';
import 'package:tflite_flutter/tflite_flutter.dart';
```

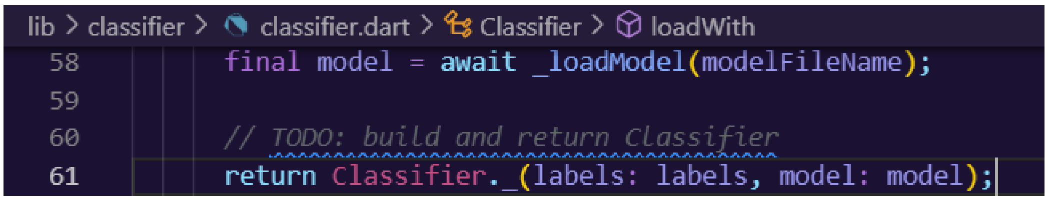
3. Next, replace // TODO: _loadModel in loadWith with the following:



```dart
    final labels = await _loadLabels(labelsFileName);

    // TODO: _loadModel
    final model = await _loadModel(modelFileName);
```

The code above loads the model file.

4. Finally, for initialization, replace // TODO: build and return Classifier in loadWith with the following:



```dart
    final model = await _loadModel(modelFileName);

    // TODO: build and return Classifier
    return Classifier._(labels: labels, model: model);
```

That builds the Classifier instance, which PlantRecogniser uses to recognize images the user provides.

Implementing TensorFlow Prediction

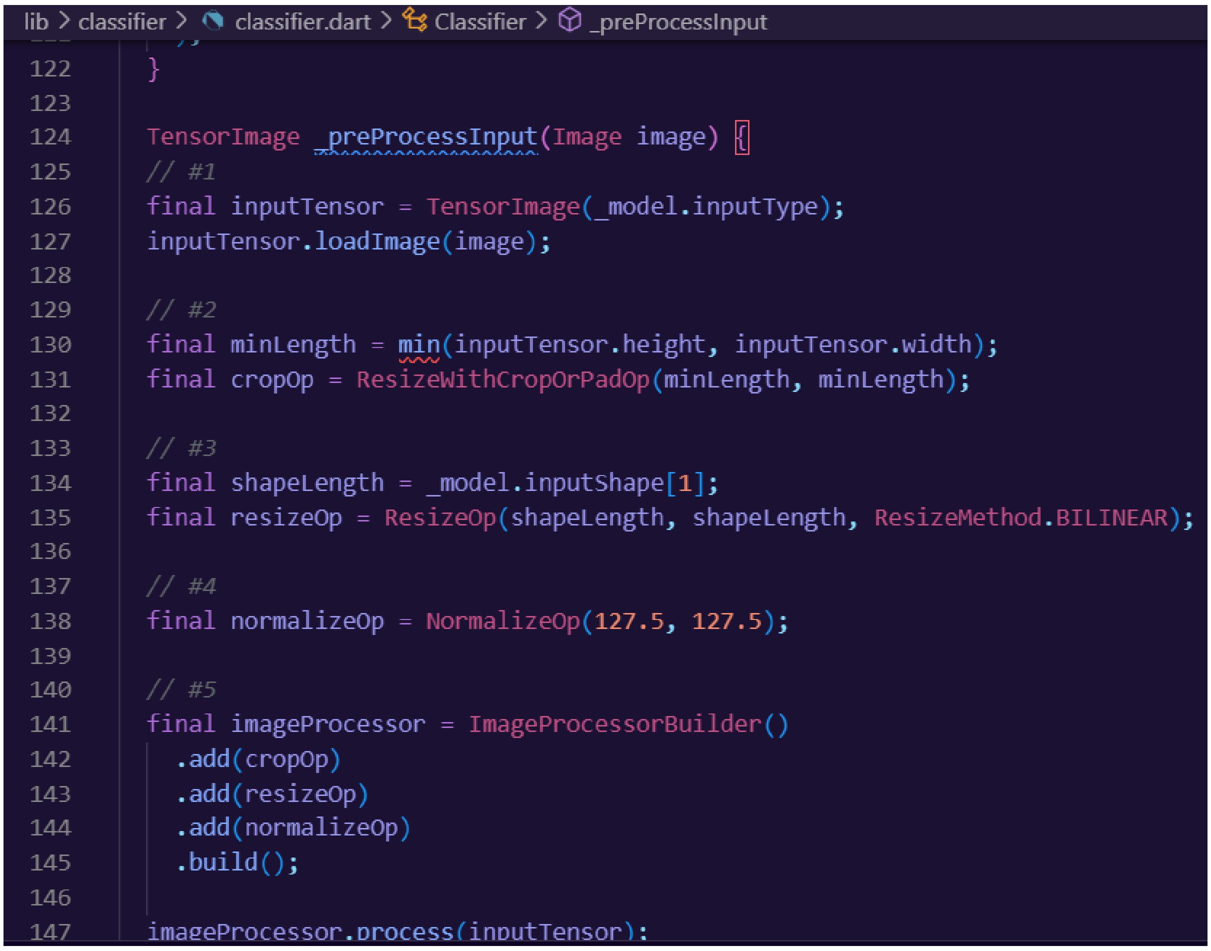
Before doing any prediction, we need to prepare the input.

We will write a method to convert the Flutter Image object to TensorImage, the tensor structure used by TensorFlow for images. We also need to modify the image to fit the required shape of the model.

Pre-Processing Image Data

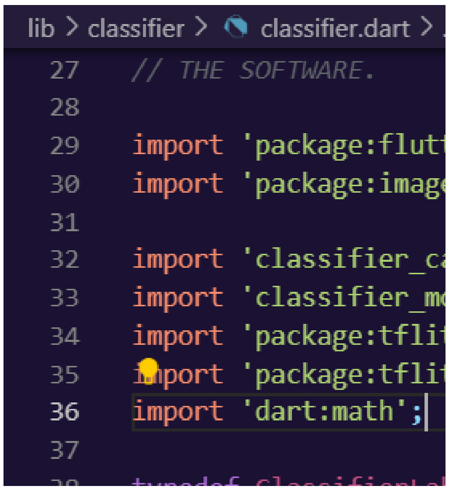
With the help of tflite_flutter_helper, image processing is simple because the library provides several functions you can pull in to handle image reshaping.

1. Add the _preProcessInput method to lib/classifier/classifier.dart:


lib > classifier > classifier.dart > Classifier > _preProcessInput

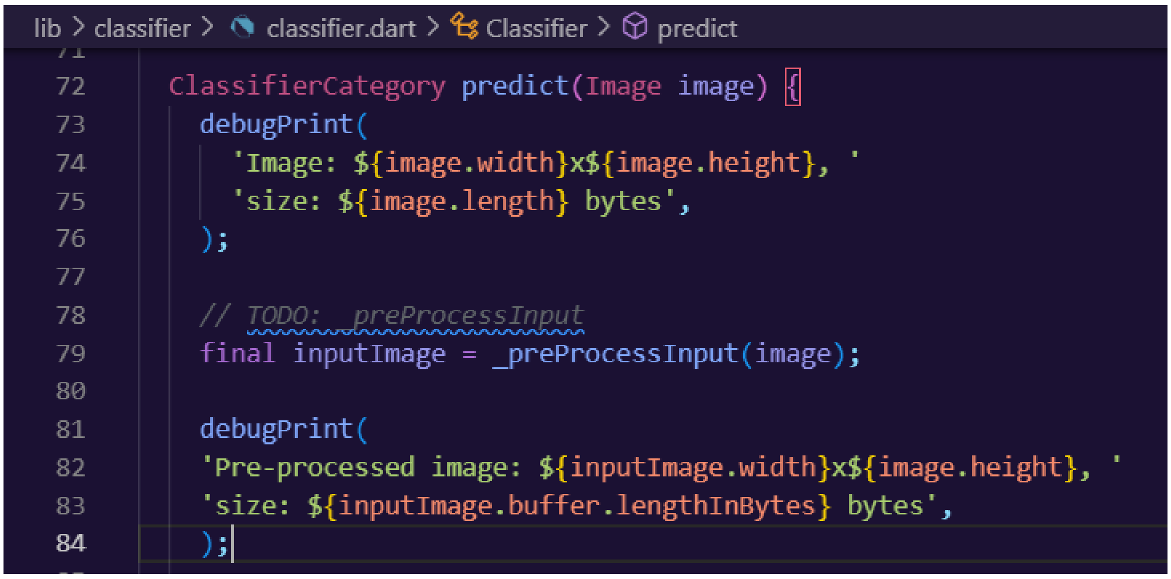
```
  }

  TensorImage _preProcessInput(Image image) {
  // #1
  final inputTensor = TensorImage(_model.inputType);
  inputTensor.loadImage(image);

  // #2
  final minLength = min(inputTensor.height, inputTensor.width);
  final cropOp = ResizeWithCropOrPadOp(minLength, minLength);

  // #3
  final shapeLength = _model.inputShape[1];
  final resizeOp = ResizeOp(shapeLength, shapeLength, ResizeMethod.BILINEAR);

  // #4
  final normalizeOp = NormalizeOp(127.5, 127.5);

  // #5
  final imageProcessor = ImageProcessorBuilder()
    .add(cropOp)
    .add(resizeOp)
    .add(normalizeOp)
    .build();

  imageProcessor.process(inputTensor);
```

You have to import dart:math at the top to use the min function.


lib > classifier > classifier.dart >

```
// THE SOFTWARE.

import 'package:flutt
import 'package:image

import 'classifier_ca
import 'classifier_mo
import 'package:tflit
import 'package:tflit
import 'dart:math';
```

2. Then, invoke the method inside predict(...) at // TODO: _preProcessInput:

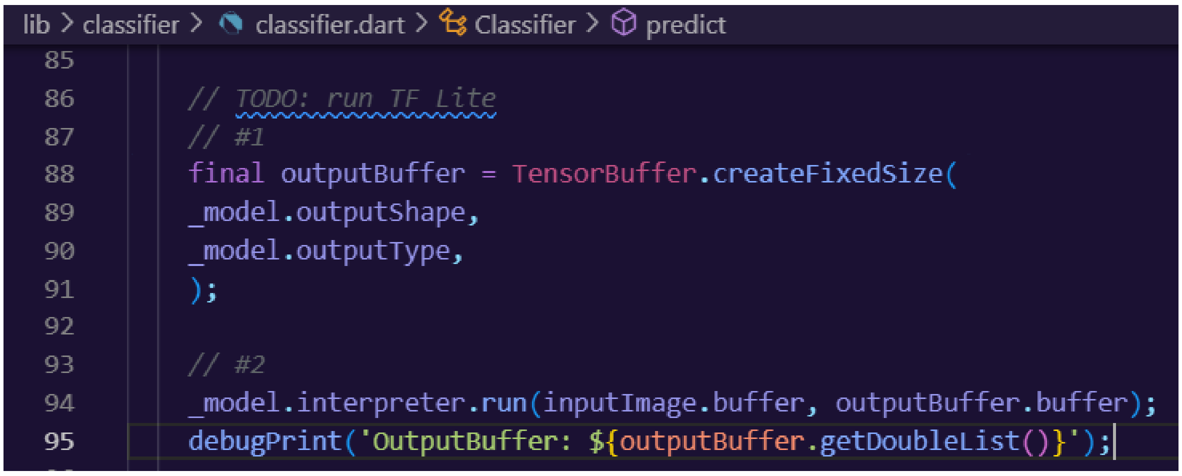


lib > classifier > classifier.dart > Classifier > predict

```dart
  ClassifierCategory predict(Image image) {
    debugPrint(
      'Image: ${image.width}x${image.height}, '
      'size: ${image.length} bytes',
    );

    // TODO: _preProcessInput
    final inputImage = _preProcessInput(image);

    debugPrint(
    'Pre-processed image: ${inputImage.width}x${image.height}, '
    'size: ${inputImage.buffer.lengthInBytes} bytes',
    );
```

Running the Prediction

1. Add the following code at // TODO: run TF Lite to run the prediction:

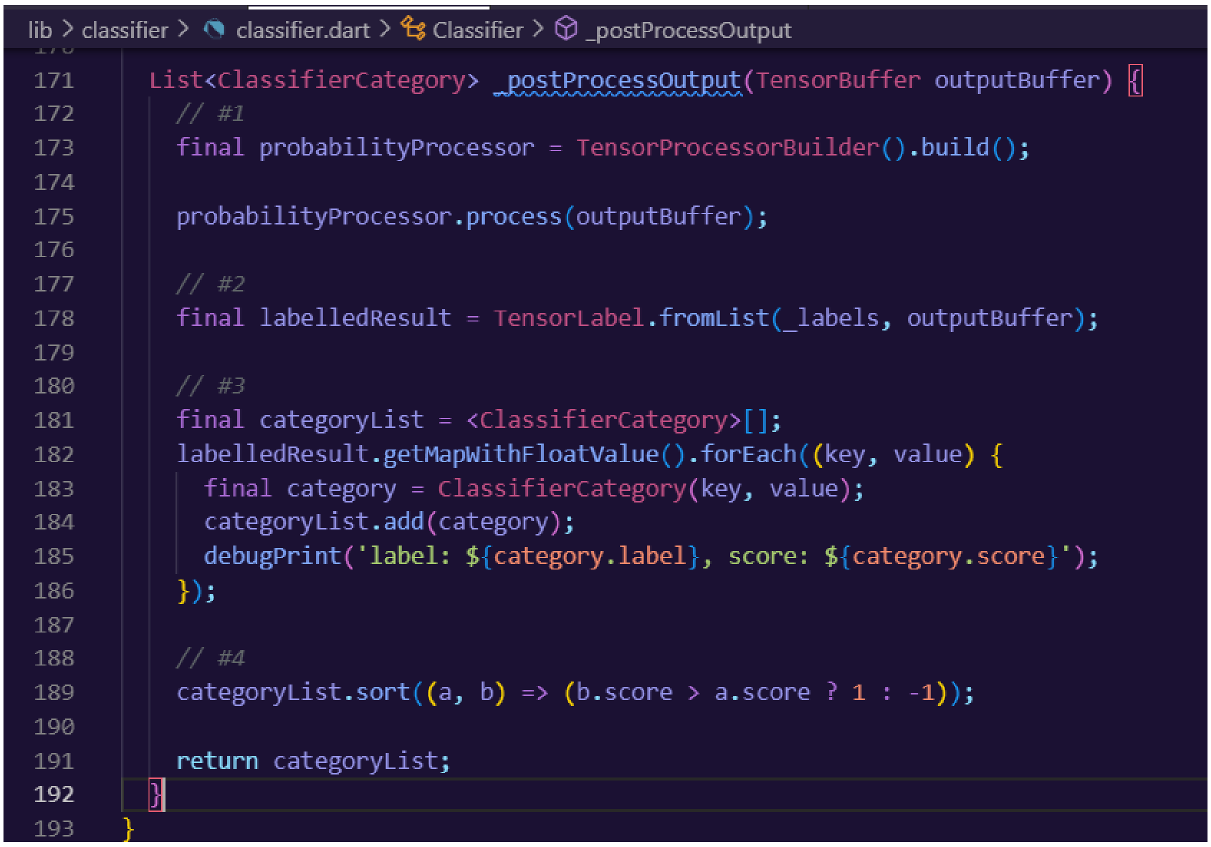


lib > classifier > classifier.dart > Classifier > predict

```dart

    // TODO: run TF Lite
    // #1
    final outputBuffer = TensorBuffer.createFixedSize(
    _model.outputShape,
    _model.outputType,
    );

    // #2
    _model.interpreter.run(inputImage.buffer, outputBuffer.buffer);
    debugPrint('OutputBuffer: ${outputBuffer.getDoubleList()}');
```

Post-Processing the Output Result

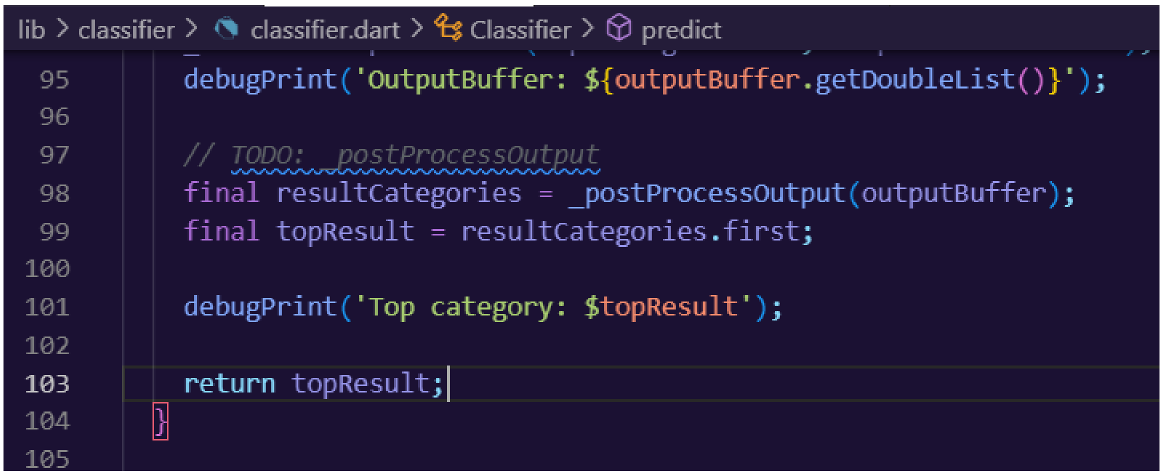
1. Add the following method to lib/classifier/classifier.dart:



lib > classifier > classifier.dart > Classifier > _postProcessOutput

```dart
  List<ClassifierCategory> _postProcessOutput(TensorBuffer outputBuffer) {
    // #1
    final probabilityProcessor = TensorProcessorBuilder().build();

    probabilityProcessor.process(outputBuffer);

    // #2
    final labelledResult = TensorLabel.fromList(_labels, outputBuffer);

    // #3
    final categoryList = <ClassifierCategory>[];
    labelledResult.getMapWithFloatValue().forEach((key, value) {
      final category = ClassifierCategory(key, value);
      categoryList.add(category);
      debugPrint('label: ${category.label}, score: ${category.score}');
    });

    // #4
    categoryList.sort((a, b) => (b.score > a.score ? 1 : -1));

    return categoryList;
  }
}
```

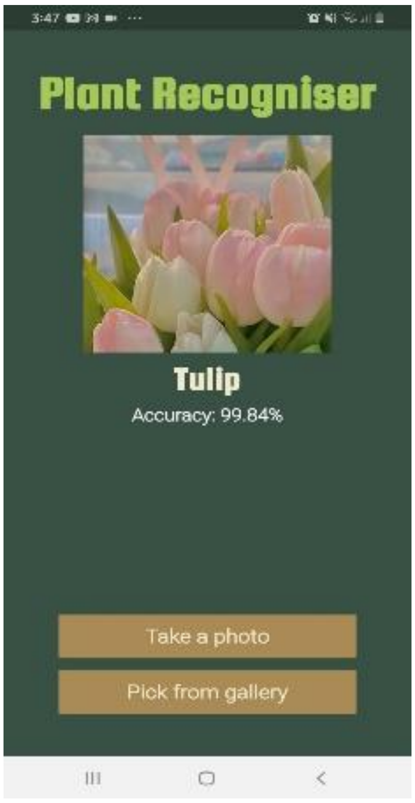
2. Now you just need to invoke _postProcessOutput() for the prediction. Update predict(...) so that it looks like the following:

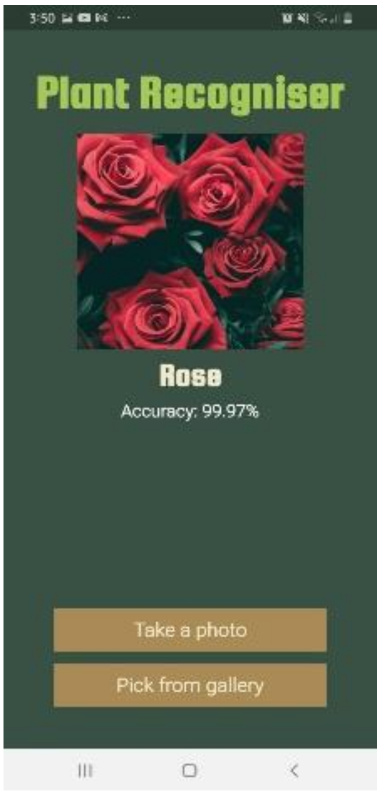


lib > classifier > classifier.dart > Classifier > predict

```dart
    debugPrint('OutputBuffer: ${outputBuffer.getDoubleList()}');

    // TODO: _postProcessOutput
    final resultCategories = _postProcessOutput(outputBuffer);
    final topResult = resultCategories.first;

    debugPrint('Top category: $topResult');

    return topResult;
  }
```

3. Build and run. Upload an image and see it correctly predicts the plant:





Conclusion

To sum up, this tutorial was a great starting point for us a beginner in Machine Learning implementation. We have learnt different important point in a very clear and concise explanation. To sum up, in this task, we have learnt how to:

- Use machine learning in a mobile app.
- Train a model using Teachable Machine.
- Integrate and use TensorFlow Lite with the tflite_flutter package.
- Build a mobile app to recognize plants by image.