RAKOTONDRAMANANA A.A. LAFATRA

INFO 3 – UNIVERSITE DES MASCAREIGNES

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

Task-1 TensorFlow Lite Tutorial for Flutter: Image Classification

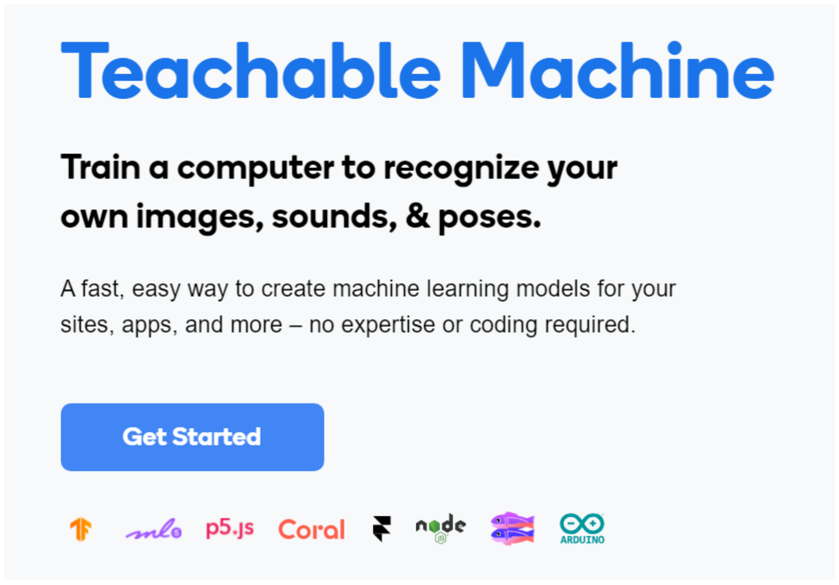
In this task, we will delve into the utilization of TensorFlow Lite within the Flutter framework. This involves training a machine learning model using Teachable Machine and seamlessly integrating the results into a Flutter mobile app.

Getting Started: To initiate this process, we will utilize a provided starter project as our foundation. While the project currently allows users to select or drag-and-drop images into the app, it lacks image recognition functionality. In the upcoming sections, we will employ TensorFlow Lite to address this limitation.

Building a Model with Teachable Machine: The starter project already incorporates a pre-trained model (model_unquant.tflite) and classification labels stored in the labels.txt file. However, our focus will be on understanding and engaging in the training process itself.

Preparing the Dataset - Training the Model: Training is the pivotal stage where the computer acquires knowledge from data and formulates rules. Our exploration will involve learning how to train the model using Teachable Machine.

1. To commence, visit https://teachablemachine.withgoogle.com and click on "Get Started" to access the training tool.

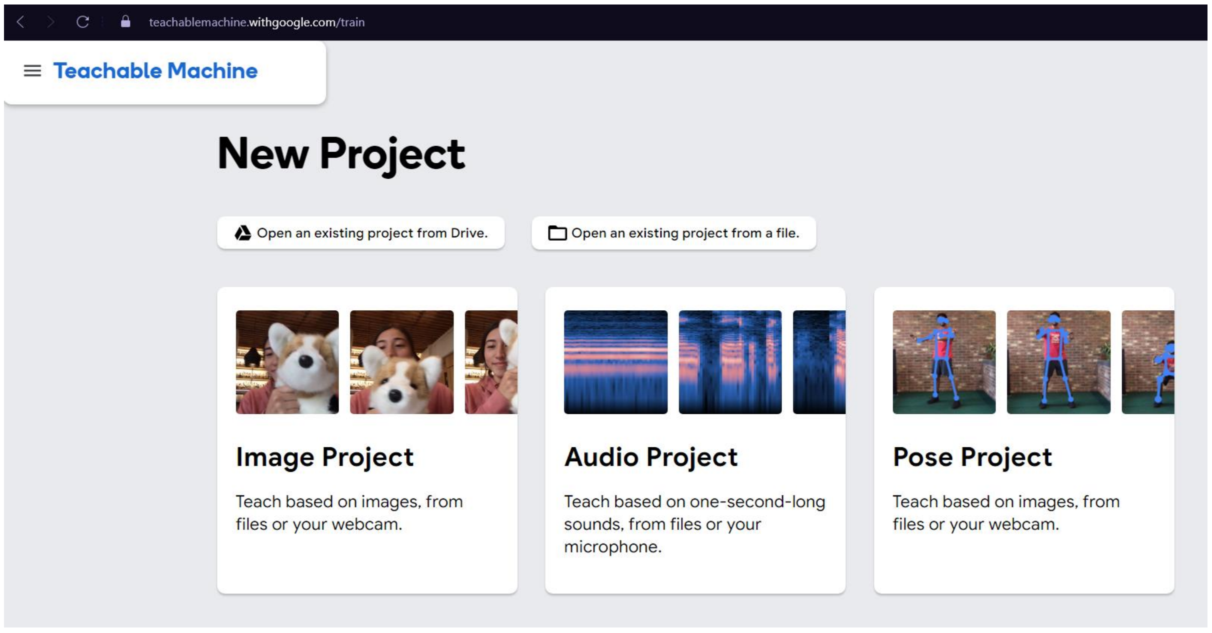


Teachable Machine

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



teachablemachine.withgoogle.com/train
Teachable Machine

New Project

Open an existing project from Drive.
Open an existing project from a file.

Image Project

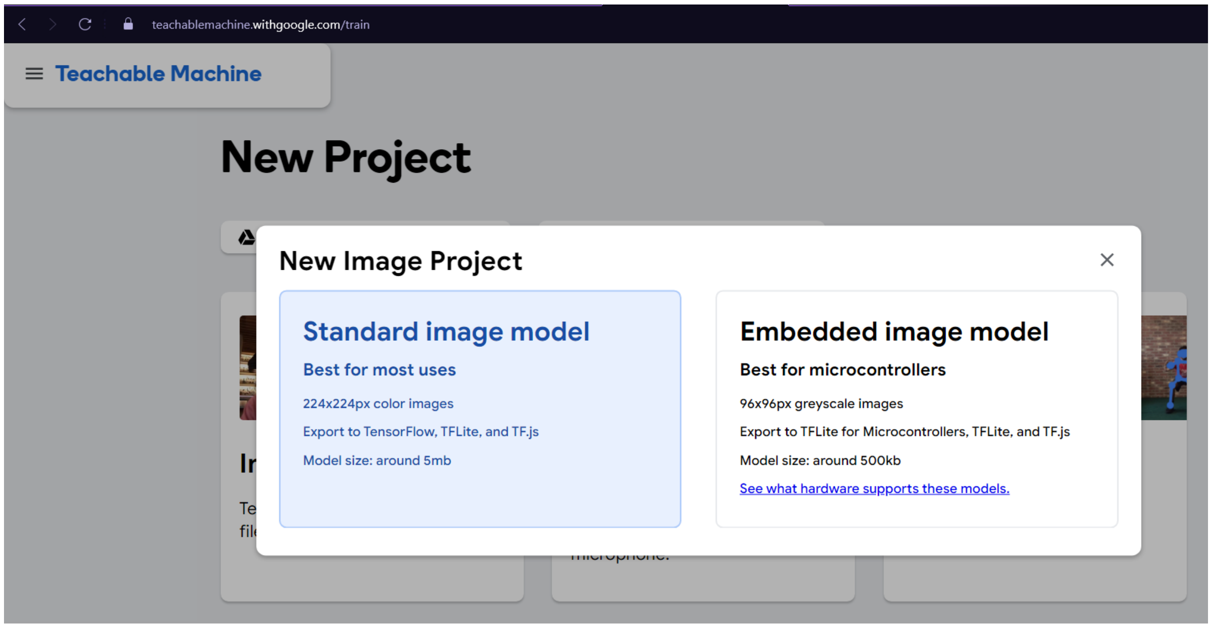
Teach based on images, from files or your webcam.

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.


teachablemachine.withgoogle.com/train

Teachable Machine

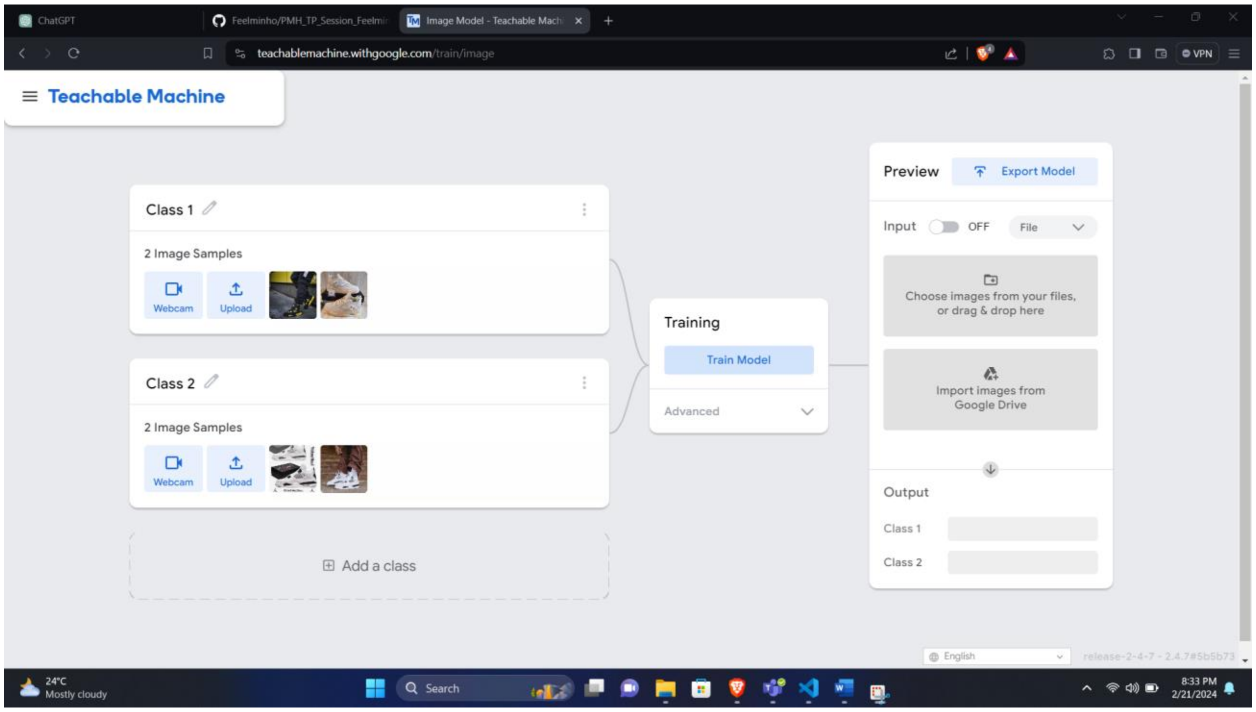
New Project

New Image Project

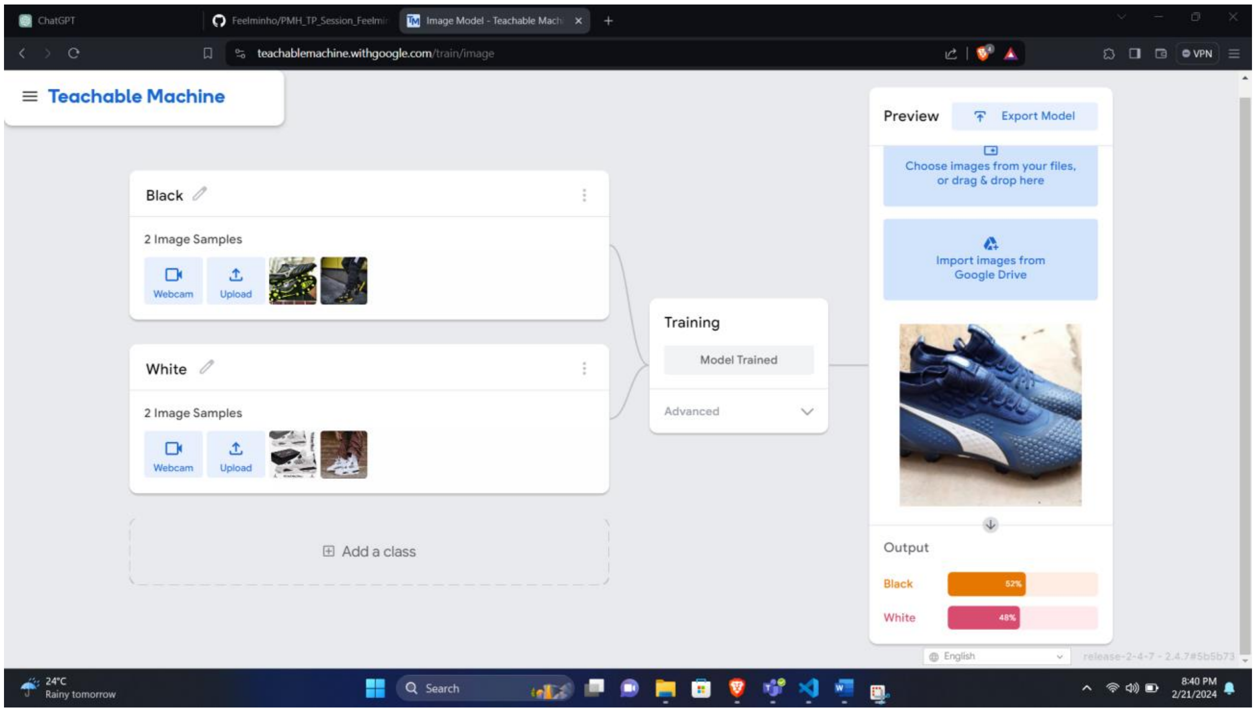
Standard image model
Best for most uses
224x224px color images
Export to TensorFlow, TFLite, and TF.js
Model size: around 5mb

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.

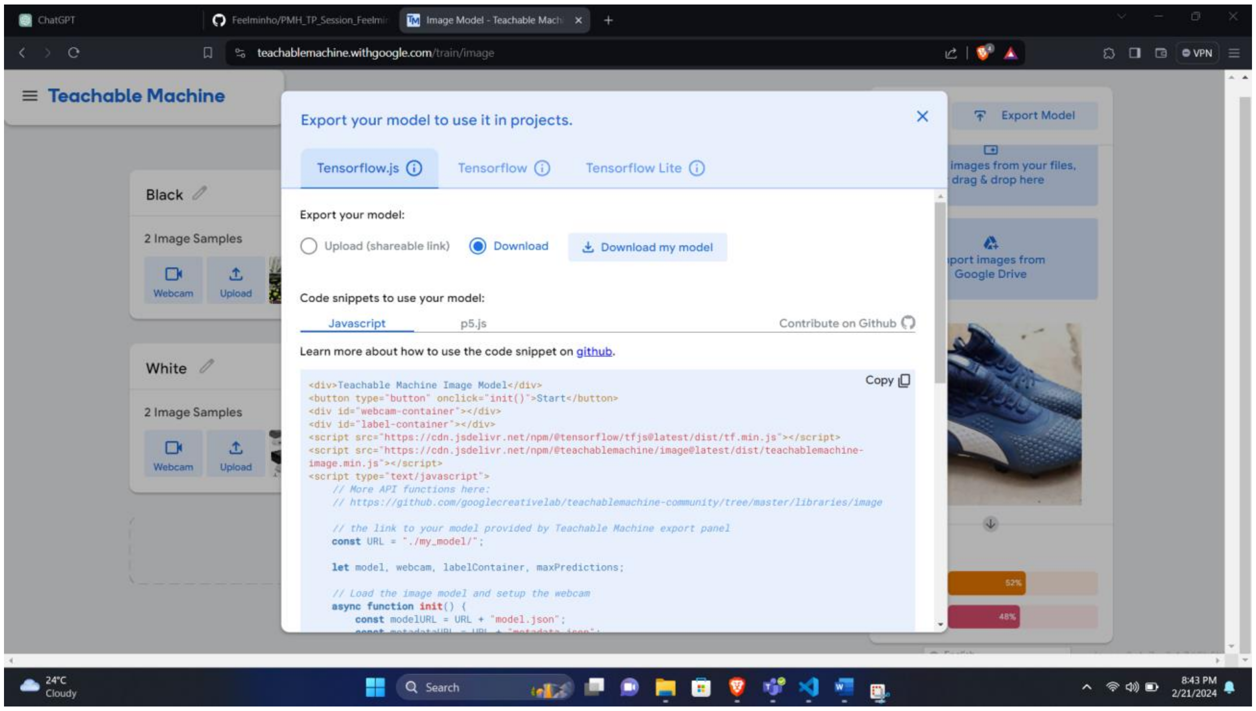
3. Next, add the training samples by clicking Upload under each class. Then, drag the folder of the appropriate plant type from the samples folder to the Choose images from your files … panel.



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



5. 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.
6. 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: Upload (shareable link) [✓ Download]

Code snippets to use your model:

Learn more about how to use the code snippet on github.

```
<div>Teachable Machine Image Model</div>
<button type="button" onclick="init()">Start</button>
<div id="webcam-container"></div>
<div id="label-container"></div>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest/dist/tf.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@teachablemachine/image@latest/dist/teachablemachine-
image.min.js"></script>
<script type="text/javascript">
    // More API functions here:
    // https://github.com/googlecreativelab/teachablemachine-community/tree/master/libraries/image

    // the link to your model provided by Teachable Machine export panel
    const URL = "./my_model/";

    let model, webcam, labelContainer, maxPredictions;

    // Load the image model and setup the webcam
    async function init() {
        const modelURL = URL + "model.json";
```

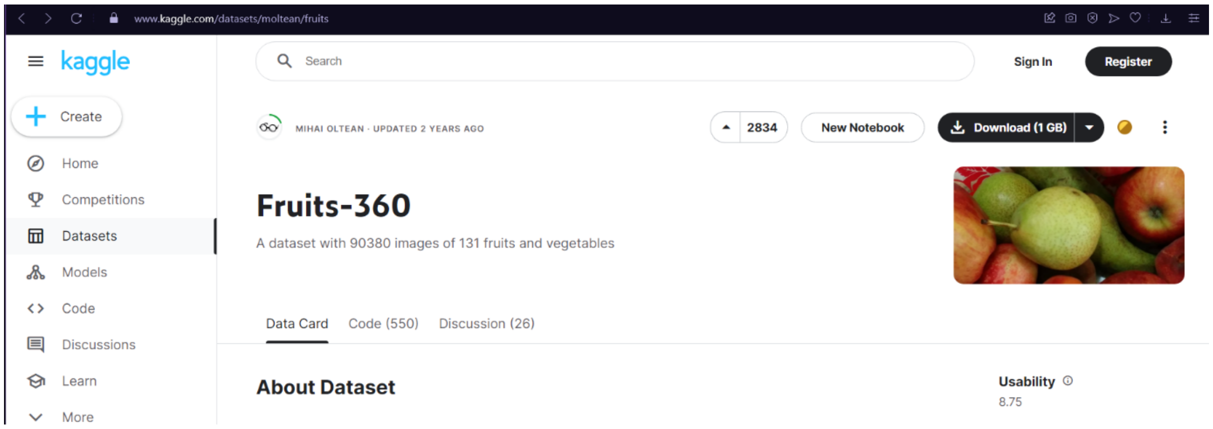
Task-2 Create an image classification Flutter application:

In this Task, we will implement an image classification Flutter application that will recognize/identify fruits and vegetable images.

To create this application, we will build a model using Teachable Machine like we have just done in the previous task. This time we will use the Fruits-360 datasets from Kaggle to train the model.

Building a Model with Teachable Machine

1. We will first download the dataset from Kaggle:



www.kaggle.com/datasets/moltean/fruits

MIHAI OLTEAN · UPDATED 2 YEARS AGO

Fruits-360

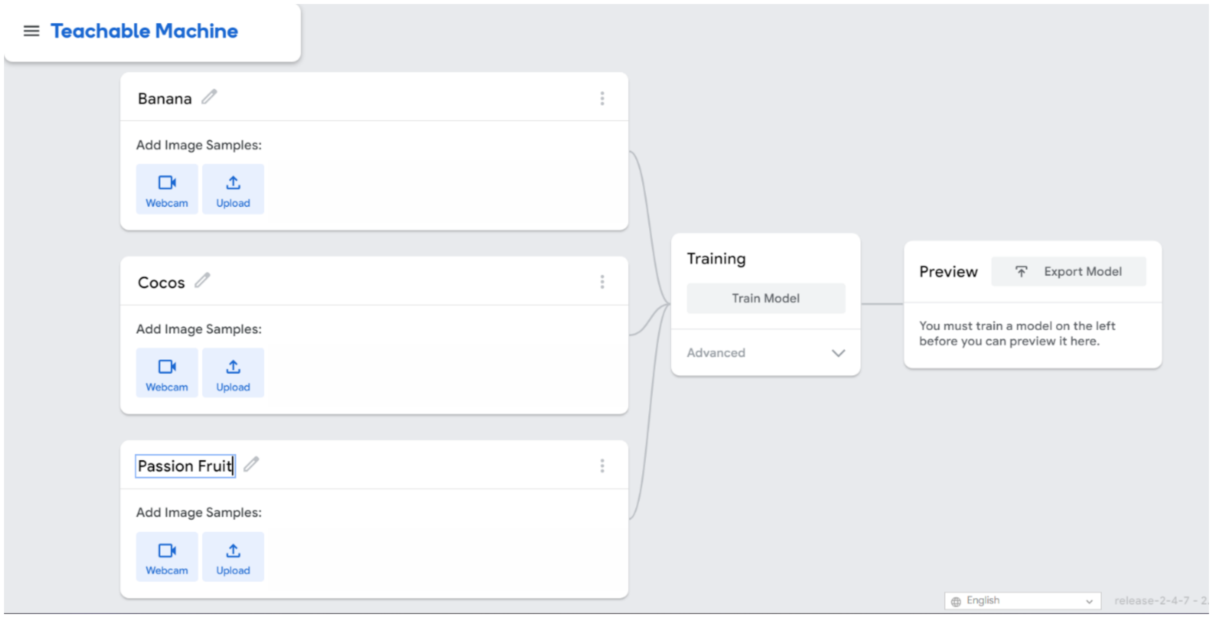
A dataset with 90380 images of 131 fruits and vegetables

About Dataset

Usability 8.75

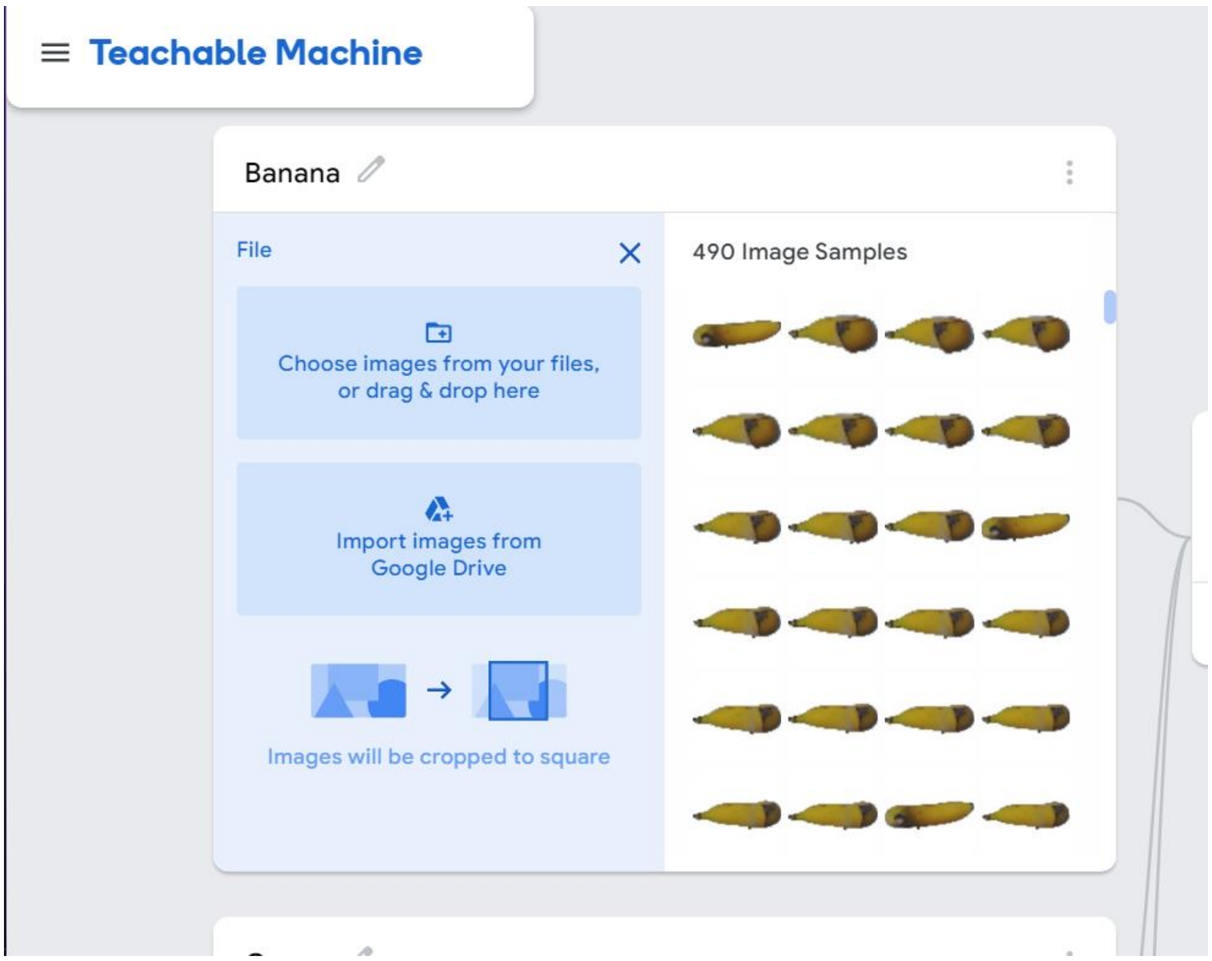
2. Next, we will create a new Image Project in Teachable machine and choose Standard Image Model when prompted.

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

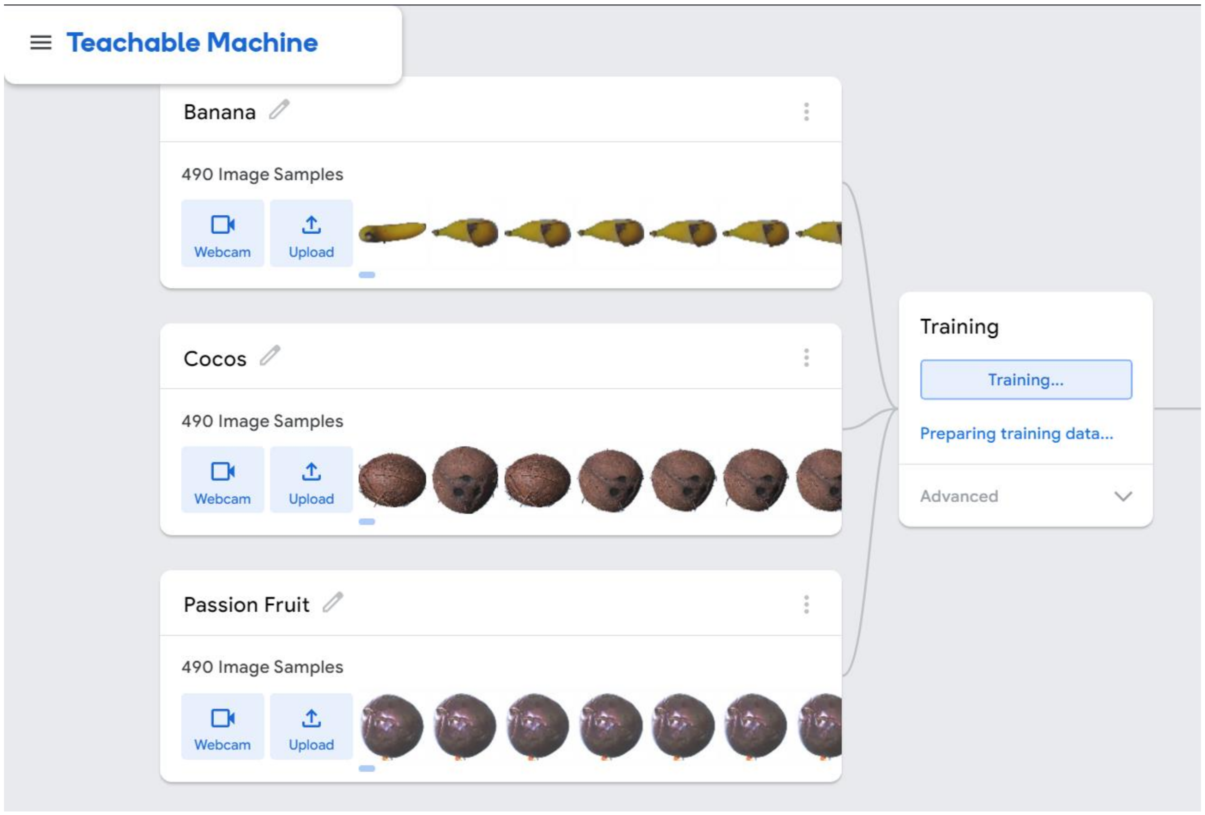


You must train a model on the left before you can preview it here.

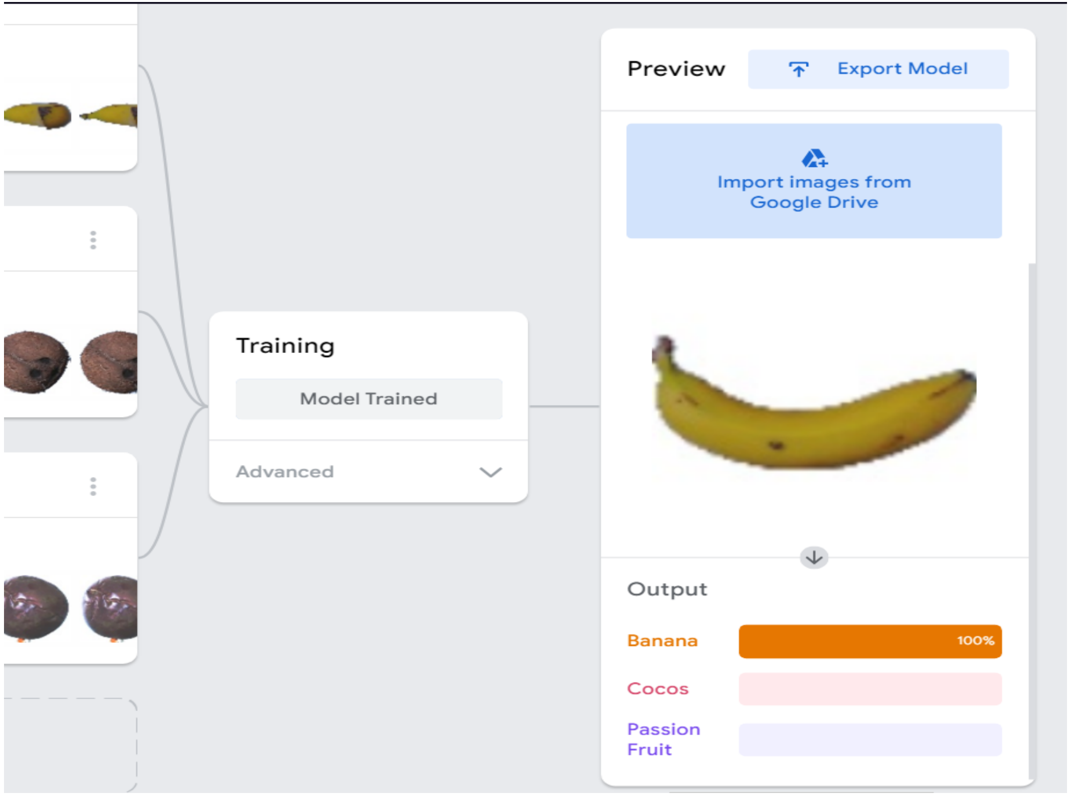
4. Next, add the training samples by clicking Upload under each class. Then, drag the folder of the appropriate plant type from the samples folder to the Choose images from your files panel.


Teachable Machine
Banana
File
Choose images from your files, or drag & drop here
Import images from Google Drive
Images will be cropped to square
490 Image Samples

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



6. After the training completes, test the model with a fruit image. Use the images in the test folder, like so:

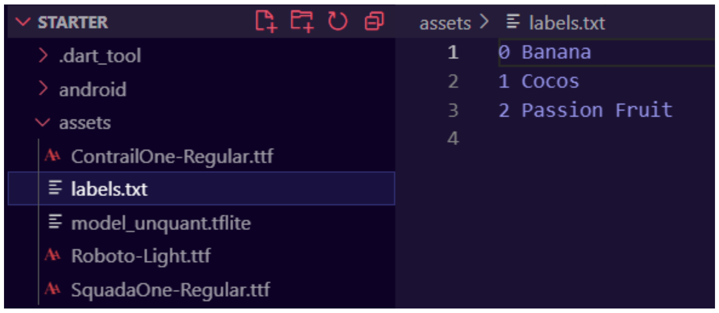


7. Finally, export the model by clicking Export Model on the Preview panel. In the dialog, choose TensorFlow Lite. That's because the target platform is mobile.
8. After you have the model file converted_tflite.zip in hand, decompress it and copy labels.txt and model_unquant.tflite to the ./assets folder in the starter project.

Implement the flutter app

We will try to modify the project created in Task 1 to recognize the fruit instead of plant.

1. We will first replace the machine learning model – model_unquant.tflite and the classification labels — labels.txt in the project plant recognition by those we got from the Teachable Machine platform.



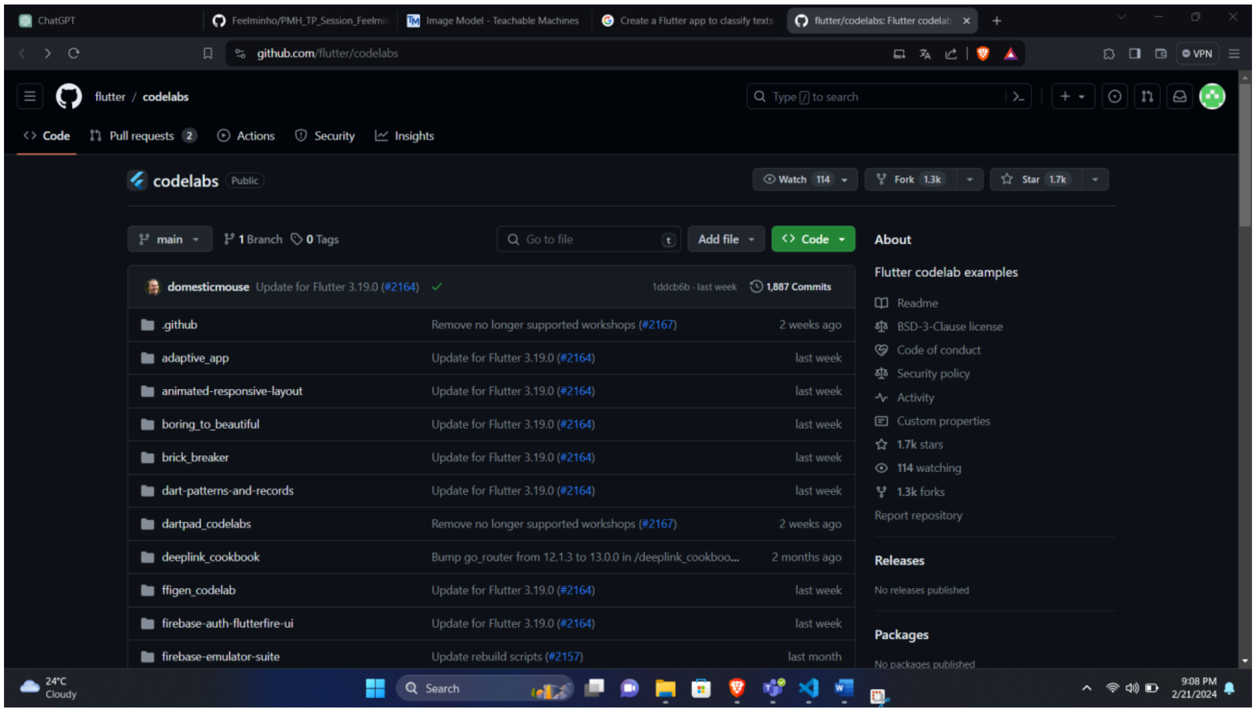
```
0 Banana
1 Cocos
2 Passion Fruit

```

Task-3: Create a Flutter app to classify texts:

This documentation guides you through executing a text classification inference from a Flutter application using TensorFlow Serving via both REST and gRPC protocols.

To obtain the code for this codelab, follow these steps:

1. Visit the GitHub repository dedicated to this codelab.
2. Click on the "Code" button and select "Download ZIP" to acquire all the necessary code files for this tutorial.



github.com/flutter/codelabs

flutter / codelabs

codelabs Public

main · 1 Branch · 0 Tags

domesticmouse Update for Flutter 3.19.0 (#2164) · 1ddcb6b · last week · 1,887 Commits

.github	Remove no longer supported workshops (#2167)	2 weeks ago
adaptive_app	Update for Flutter 3.19.0 (#2164)	last week
animated-responsive-layout	Update for Flutter 3.19.0 (#2164)	last week
boring_to_beautiful	Update for Flutter 3.19.0 (#2164)	last week
brick_breaker	Update for Flutter 3.19.0 (#2164)	last week
dart-patterns-and-records	Update for Flutter 3.19.0 (#2164)	last week
dartpad_codelabs	Remove no longer supported workshops (#2167)	2 weeks ago
deeplink_cookbook	Bump go_router from 12.1.3 to 13.0.0 in /deeplink_cookboo...	2 months ago
ffigen_codelab	Update for Flutter 3.19.0 (#2164)	last week
firebase-auth-flutterfire-ui	Update for Flutter 3.19.0 (#2164)	last week
firebase-emulator-suite	Update rebuild scripts (#2157)	last month

About

Flutter codelab examples

Readme · BSD-3-Clause license · Code of conduct · Security policy · Activity · Custom properties · 1.7k stars · 114 watching · 1.3k forks

Report repository

Releases

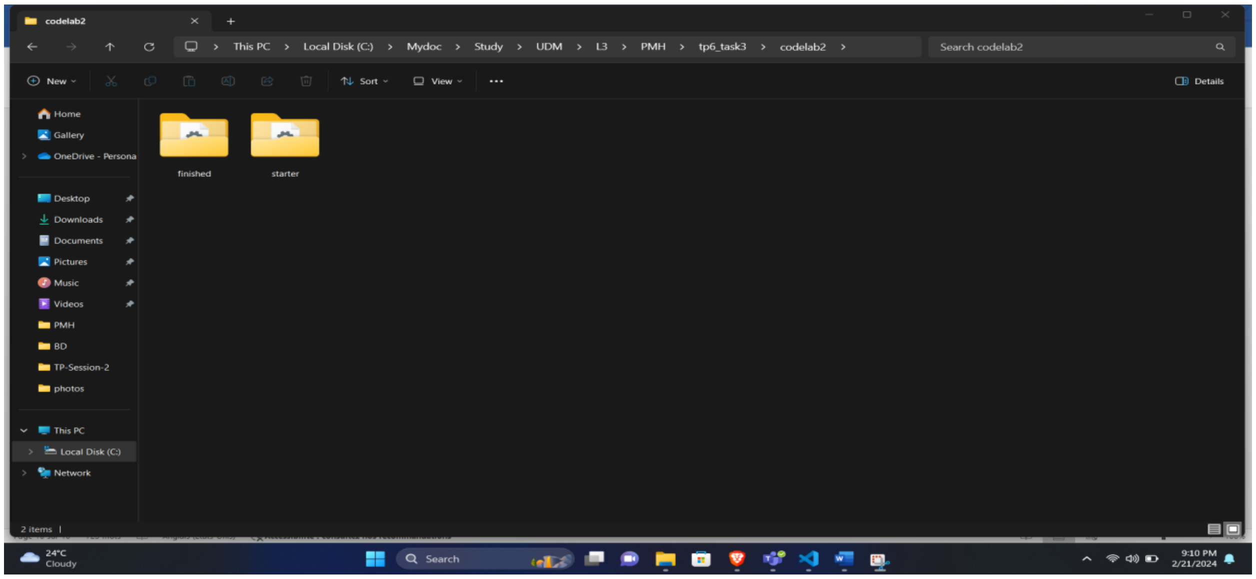
No releases published

Packages

No packages published

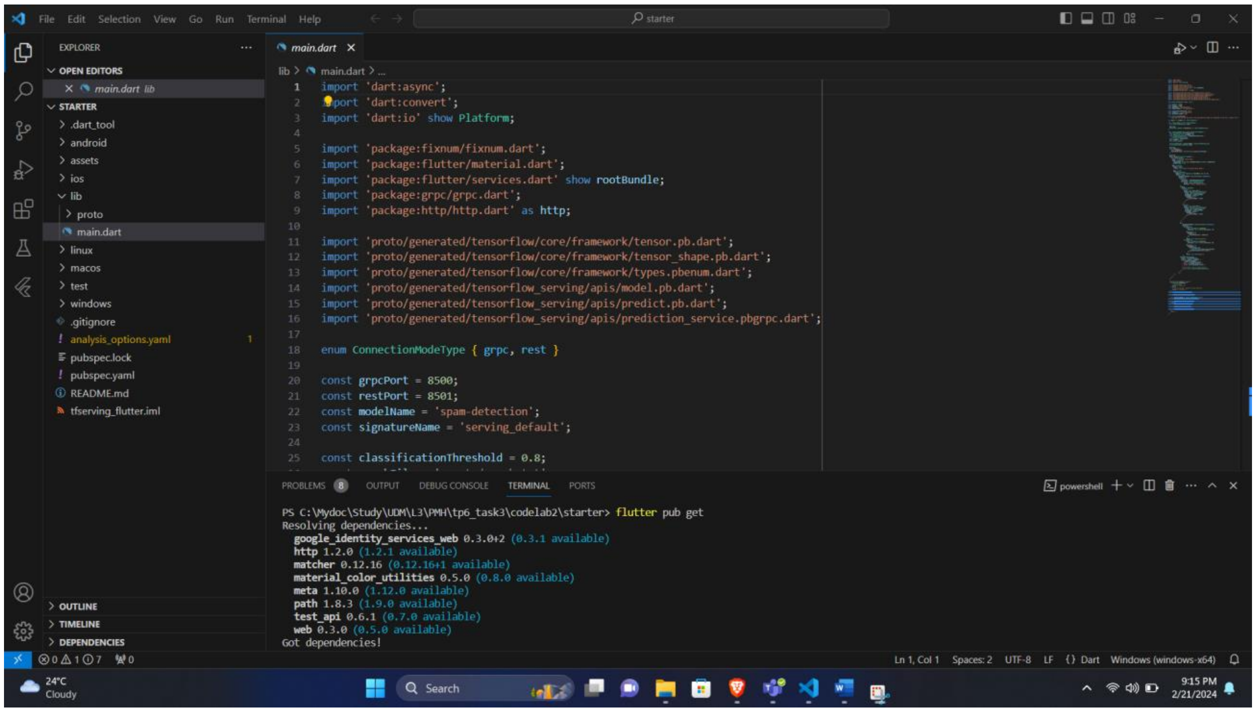
In this codelab, you will exclusively utilize the files located in the "tfserving-flutter/codelab2" subdirectory within the repository. This subdirectory comprises two main folders:

- **Starter Folder:** This contains the initial code that serves as the foundation for your work in this codelab.
- **Finished Folder:** Here, you can find the fully developed code representing the completed sample app.


codelab2
This PC > Local Disk (C:) > Mydoc > Study > UDM > L3 > PMH > tp6_task3 > codelab2
Search codelab2
finished
starter
2 items

3. Retrieve the project dependencies by following these steps:

- Open Visual Studio Code (VS Code).
- Navigate to "File" > "Open Folder."
- Choose the starter folder from the previously downloaded source code. Run this command: flutter pub get

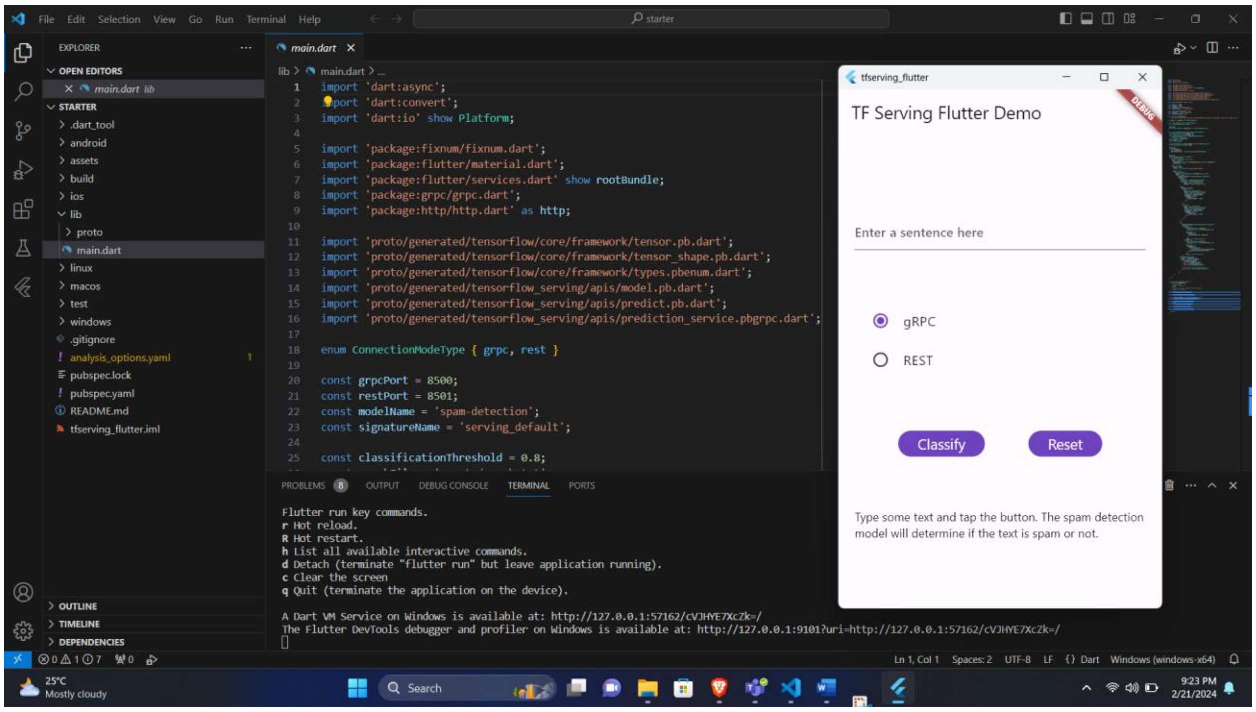


```dart
import 'dart:async';
import 'dart:convert';
import 'dart:io' show Platform;

import 'package:fixnum/fixnum.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart' show rootBundle;
import 'package:grpc/grpc.dart';
import 'package:http/http.dart' as http;

import 'proto/generated/tensorflow/core/framework/tensor.pb.dart';
import 'proto/generated/tensorflow/core/framework/tensor_shape.pb.dart';
import 'proto/generated/tensorflow/core/framework/types.pbenum.dart';
import 'proto/generated/tensorflow_serving/apis/model.pb.dart';
import 'proto/generated/tensorflow_serving/apis/predict.pb.dart';
import 'proto/generated/tensorflow_serving/apis/prediction_service.pbgrpc.dart';

enum ConnectionModeType { grpc, rest }

const grpcPort = 8500;
const restPort = 8501;
const modelName = 'spam-detection';
const signatureName = 'serving_default';

const classificationThreshold = 0.8;
```

```
PS C:\Mydoc\Study\UDM\L3\PMH\tp6_task3\codelab2\starter> flutter pub get
Resolving dependencies...
  google_identity_services_web 0.3.0+2 (0.3.1 available)
  http 1.2.0 (1.2.1 available)
  matcher 0.12.16 (0.12.16+1 available)
  material_color_utilities 0.5.0 (0.8.0 available)
  meta 1.10.0 (1.12.0 available)
  path 1.8.3 (1.9.0 available)
  test_api 0.6.1 (0.7.0 available)
  web 0.3.0 (0.5.0 available)
Got dependencies!
```

4. Execute and navigate through the application:

Once launched, the app will appear on your Android Emulator or iOS Simulator. The user interface is intuitive, featuring a text field allowing users to input text. Users have the option to transmit data to the backend using either REST or gRPC. The backend employs a TensorFlow model to conduct text classification on the processed input, subsequently relaying the classification outcome to the client app. The app, in turn, updates the UI to reflect the result.



```
import 'dart:async';
import 'dart:convert';
import 'dart:io' show Platform;

import 'package:fixnum/fixnum.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart' show rootBundle;
import 'package:grpc/grpc.dart';
import 'package:http/http.dart' as http;

import 'proto/generated/tensorflow/core/framework/tensor.pb.dart';
import 'proto/generated/tensorflow/core/framework/tensor_shape.pb.dart';
import 'proto/generated/tensorflow/core/framework/types.pbenum.dart';
import 'proto/generated/tensorflow_serving/apis/model.pb.dart';
import 'proto/generated/tensorflow_serving/apis/predict.pb.dart';
import 'proto/generated/tensorflow_serving/apis/prediction_service.pbgrpc.dart';

enum ConnectionModeType { grpc, rest }

const grpcPort = 8500;
const restPort = 8501;
const modelName = 'spam-detection';
const signatureName = 'serving_default';

const classificationThreshold = 0.8;
```

```
Flutter run key commands.
r Hot reload.
R Hot restart.
h List all available interactive commands.
d Detach (terminate "flutter run" but leave application running).
c Clear the screen
q Quit (terminate the application on the device).

A Dart VM Service on Windows is available at: http://127.0.0.1:57162/cVJHYE7XcZk=/
The Flutter DevTools debugger and profiler on Windows is available at: http://127.0.0.1:9101?uri=http://127.0.0.1:57162/cVJHYE7XcZk=/
```

Type some text and tap the button. The spam detection model will determine if the text is spam or not.

Deploy a text-classification model with TensorFlow Serving

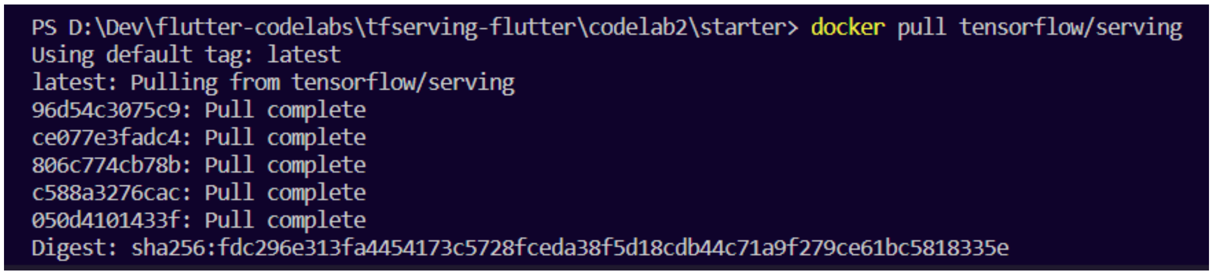
Text classification is a very common machine learning task that classifies texts into predefined categories.

In this codelab, we will deploy a pretrained model from the Train a comment-spam detection model with TensorFlow Lite Model Maker codelab with TensorFlow Serving and call the backend from our Flutter frontend to classify the input text as ***spam* or *not spam***.

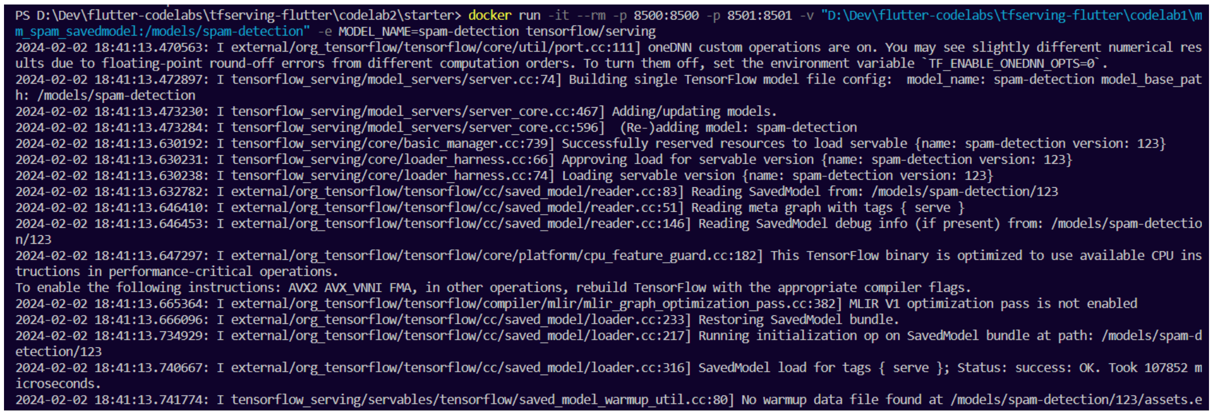
Note: A pretrained SavedModel along with the vocabulary and label files is provided in the *tfserving-flutter/codelab1/mm_spam_savedmodel* folder.

Start TensorFlow Serving

- In a terminal, start TensorFlow Serving with Docker, but replace the *PATH/TO/SAVEDMODEL* placeholder with the absolute path of the mm_spam_savedmodel folder on the computer.

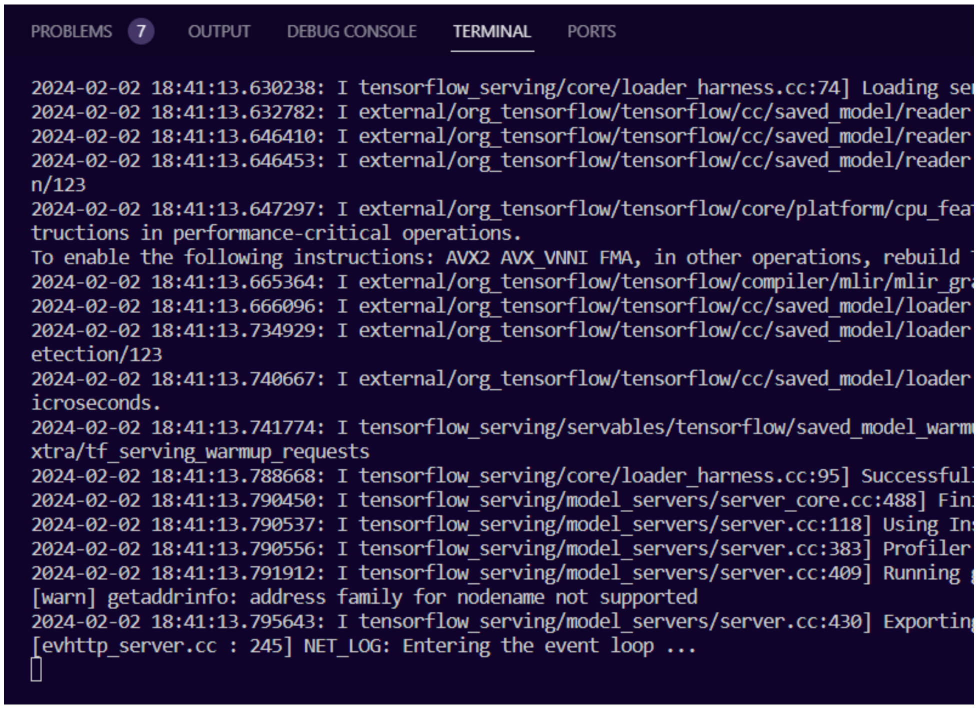


```
PS D:\Dev\flutter-codelabs\tfserving-flutter\codelab2\starter> docker pull tensorflow/serving
Using default tag: latest
latest: Pulling from tensorflow/serving
96d54c3075c9: Pull complete
ce077e3fadc4: Pull complete
806c774cb78b: Pull complete
c588a3276cac: Pull complete
050d4101433f: Pull complete
Digest: sha256:fdc296e313fa4454173c5728fceda38f5d18cdb44c71a9f279ce61bc5818335e
```



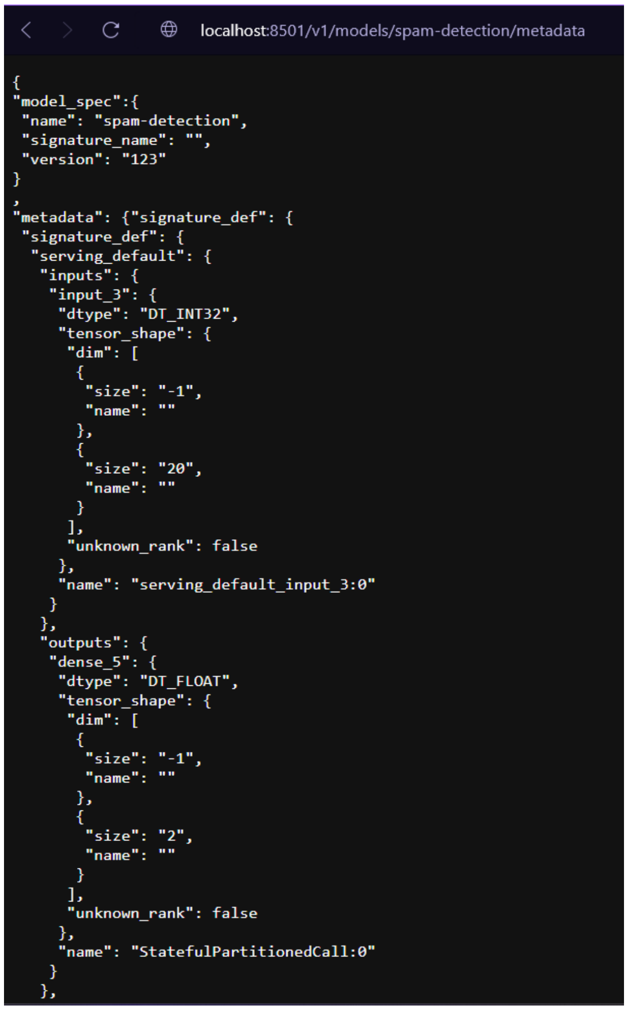
```
PS D:\Dev\flutter-codelabs\tfserving-flutter\codelab2\starter> docker run -it --rm -p 8500:8500 -p 8501:8501 -v "D:\Dev\flutter-codelabs\tfserving-flutter\codelab1\m
m_spam_savedmodel:/models/spam-detection" -e MODEL_NAME=spam-detection tensorflow/serving
2024-02-02 18:41:13.470563: I external/org_tensorflow/tensorflow/core/util/port.cc:111] oneDNN custom operations are on. You may see slightly different numerical res
ults due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-02-02 18:41:13.472897: I tensorflow_serving/model_servers/server.cc:74] Building single TensorFlow model file config:  model_name: spam-detection model_base_pat
h: /models/spam-detection
2024-02-02 18:41:13.473230: I tensorflow_serving/model_servers/server_core.cc:467] Adding/updating models.
2024-02-02 18:41:13.473284: I tensorflow_serving/model_servers/server_core.cc:596]  (Re-)adding model: spam-detection
2024-02-02 18:41:13.630192: I tensorflow_serving/core/basic_manager.cc:739] Successfully reserved resources to load servable {name: spam-detection version: 123}
2024-02-02 18:41:13.630231: I tensorflow_serving/core/loader_harness.cc:66] Approving load for servable version {name: spam-detection version: 123}
2024-02-02 18:41:13.630238: I tensorflow_serving/core/loader_harness.cc:74] Loading servable version {name: spam-detection version: 123}
2024-02-02 18:41:13.632782: I external/org_tensorflow/tensorflow/cc/saved_model/reader.cc:83] Reading SavedModel from: /models/spam-detection/123
2024-02-02 18:41:13.646410: I external/org_tensorflow/tensorflow/cc/saved_model/reader.cc:51] Reading meta graph with tags { serve }
2024-02-02 18:41:13.646453: I external/org_tensorflow/tensorflow/cc/saved_model/reader.cc:146] Reading SavedModel debug info (if present) from: /models/spam-detectio
n/123
2024-02-02 18:41:13.647297: I external/org_tensorflow/tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU ins
tructions in performance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-02-02 18:41:13.665364: I external/org_tensorflow/tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:382] MLIR V1 optimization pass is not enabled
2024-02-02 18:41:13.666096: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:233] Restoring SavedModel bundle.
2024-02-02 18:41:13.734929: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:217] Running initialization op on SavedModel bundle at path: /models/spam-d
etection/123
2024-02-02 18:41:13.740667: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:316] SavedModel load for tags { serve }; Status: success: OK. Took 107852 m
icroseconds.
2024-02-02 18:41:13.741774: I tensorflow_serving/servables/tensorflow/saved_model_warmup_util.cc:80] No warmup data file found at /models/spam-detection/123/assets.e
```

Docker automatically downloads the TensorFlow Serving image first, which takes a minute. Afterward, TensorFlow Serving should start. The log should look like this code snippet:



```
2024-02-02 18:41:13.630238: I tensorflow_serving/core/loader_harness.cc:74] Loading se
2024-02-02 18:41:13.632782: I external/org_tensorflow/tensorflow/cc/saved_model/reader
2024-02-02 18:41:13.646410: I external/org_tensorflow/tensorflow/cc/saved_model/reader
2024-02-02 18:41:13.646453: I external/org_tensorflow/tensorflow/cc/saved_model/reader
n/123
2024-02-02 18:41:13.647297: I external/org_tensorflow/tensorflow/core/platform/cpu_fea
tructions in performance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild
2024-02-02 18:41:13.665364: I external/org_tensorflow/tensorflow/compiler/mlir/mlir_gr
2024-02-02 18:41:13.666096: I external/org_tensorflow/tensorflow/cc/saved_model/loader
2024-02-02 18:41:13.734929: I external/org_tensorflow/tensorflow/cc/saved_model/loader
etection/123
2024-02-02 18:41:13.740667: I external/org_tensorflow/tensorflow/cc/saved_model/loader
icroseconds.
2024-02-02 18:41:13.741774: I tensorflow_serving/servables/tensorflow/saved_model_warm
xtra/tf_serving_warmup_requests
2024-02-02 18:41:13.788668: I tensorflow_serving/core/loader_harness.cc:95] Successful
2024-02-02 18:41:13.790450: I tensorflow_serving/model_servers/server_core.cc:488] Fin
2024-02-02 18:41:13.790537: I tensorflow_serving/model_servers/server.cc:118] Using In
2024-02-02 18:41:13.790556: I tensorflow_serving/model_servers/server.cc:383] Profiler
2024-02-02 18:41:13.791912: I tensorflow_serving/model_servers/server.cc:409] Running
[warn] getaddrinfo: address family for nodename not supported
2024-02-02 18:41:13.795643: I tensorflow_serving/model_servers/server.cc:430] Exportin
[evhttp_server.cc : 245] NET_LOG: Entering the event loop ...
```

After starting the TensorFlow Serving docker image, we can visit *http://localhost:8501/v1/models/spam-detection/metadata* to inspect the details of the input and output tensors.



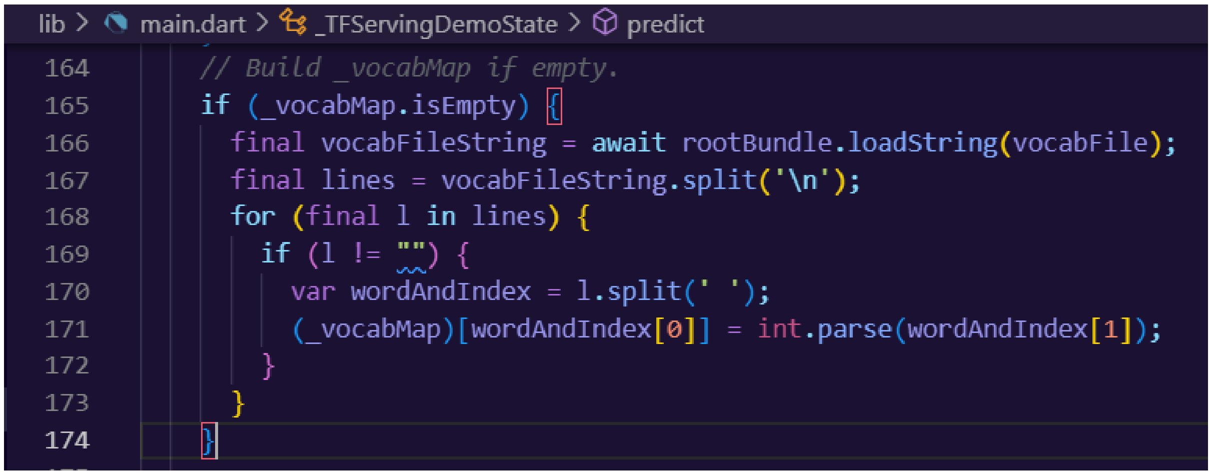
```
{
"model_spec":{
 "name": "spam-detection",
 "signature_name": "",
 "version": "123"
}
,
"metadata": {"signature_def": {
 "signature_def": {
  "serving_default": {
   "inputs": {
    "input_3": {
     "dtype": "DT_INT32",
     "tensor_shape": {
      "dim": [
       {
        "size": "-1",
        "name": ""
       },
       {
        "size": "20",
        "name": ""
       }
      ],
      "unknown_rank": false
     },
     "name": "serving_default_input_3:0"
    }
   },
   "outputs": {
    "dense_5": {
     "dtype": "DT_FLOAT",
     "tensor_shape": {
      "dim": [
       {
        "size": "-1",
        "name": ""
       },
       {
        "size": "2",
        "name": ""
       }
      ],
      "unknown_rank": false
     },
     "name": "StatefulPartitionedCall:0"
    }
   },
```

Tokenize input sentence

After preparing the backend, the next step is to tokenize the input sentence, enabling the transmission of client requests to TensorFlow Serving. By examining the input tensor of the model, it becomes apparent that it anticipates a list of 20 integer numbers instead of raw strings.

Tokenization involves mapping individual words entered in the app to a list of integers based on a predefined vocabulary dictionary before forwarding them to the backend for classification.

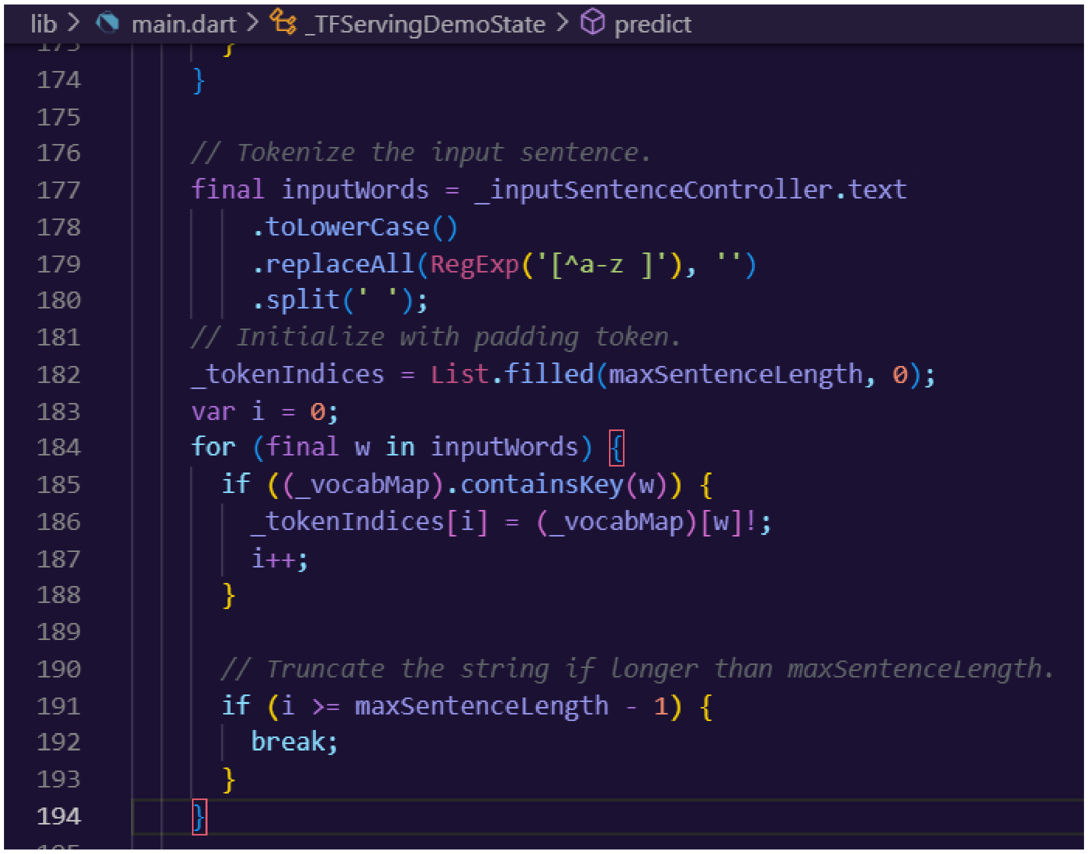
To implement this, add the following code to the **predict()** method in the **lib/main.dart** file to construct the **_vocabMap** vocabulary dictionary.



lib > main.dart > _TFServingDemoState > predict

```dart
    // Build _vocabMap if empty.
    if (_vocabMap.isEmpty) {
      final vocabFileString = await rootBundle.loadString(vocabFile);
      final lines = vocabFileString.split('\n');
      for (final l in lines) {
        if (l != "") {
          var wordAndIndex = l.split(' ');
          (_vocabMap)[wordAndIndex[0]] = int.parse(wordAndIndex[1]);
        }
      }
    }
```

Immediately after the previous code snippet, add this code to implement tokenization:



lib > main.dart > _TFServingDemoState > predict

```dart
    }

    // Tokenize the input sentence.
    final inputWords = _inputSentenceController.text
        .toLowerCase()
        .replaceAll(RegExp('[^a-z ]'), '')
        .split(' ');
    // Initialize with padding token.
    _tokenIndices = List.filled(maxSentenceLength, 0);
    var i = 0;
    for (final w in inputWords) {
      if ((_vocabMap).containsKey(w)) {
        _tokenIndices[i] = (_vocabMap)[w]!;
        i++;
      }

      // Truncate the string if longer than maxSentenceLength.
      if (i >= maxSentenceLength - 1) {
        break;
      }
    }
```

This code lowercases the sentence string, removes non-alphabet characters, and maps the words to 20 integer indices based on the vocabulary table.

Connect the Flutter app with TensorFlow Serving through REST

There are two ways to send requests to TensorFlow Serving:

- REST
- gRPC

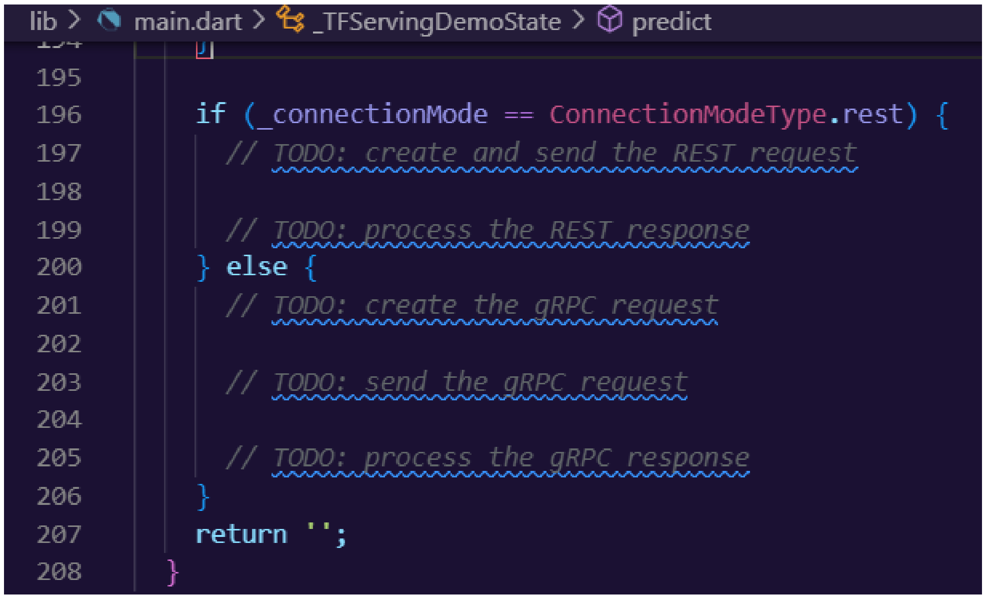
Send requests and receive responses through REST

There are three simple steps to send requests and receive responses through REST:

1. Create the REST request.
2. Send the REST request to TensorFlow Serving.
3. Extract the predicted result from the REST response and render the UI.

Create and send the REST request to TensorFlow Serving

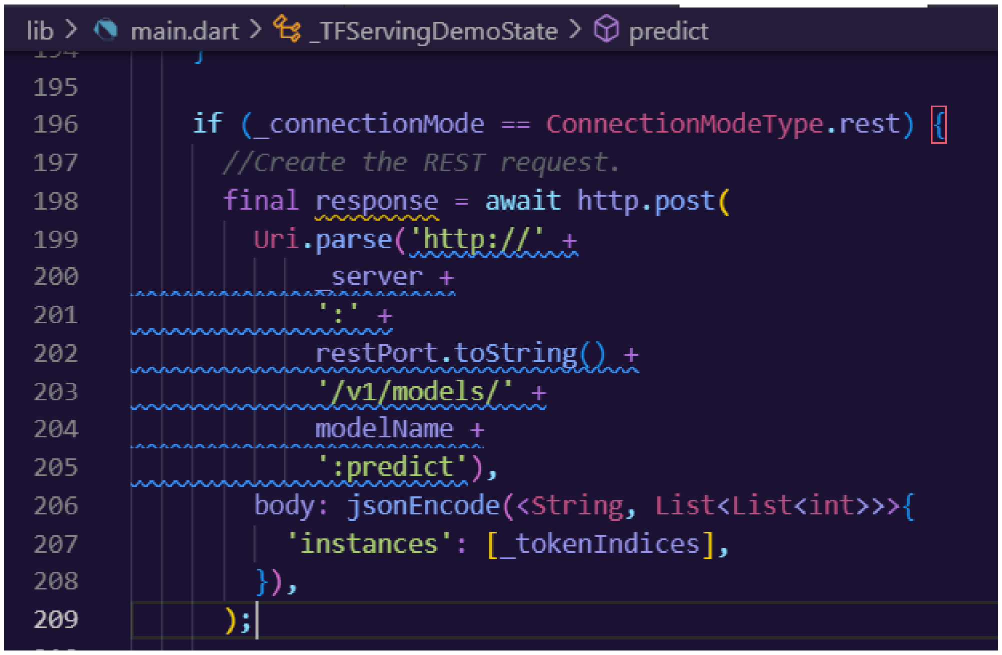
1. Right now, the predict() function doesn't send the REST request to TensorFlow Serving. You need to implement the REST branch to create a REST request:



lib > main.dart > _TFServingDemoState > predict

```
    if (_connectionMode == ConnectionModeType.rest) {
      // TODO: create and send the REST request

      // TODO: process the REST response
    } else {
      // TODO: create the gRPC request

      // TODO: send the gRPC request

      // TODO: process the gRPC response
    }
    return '';
  }
```

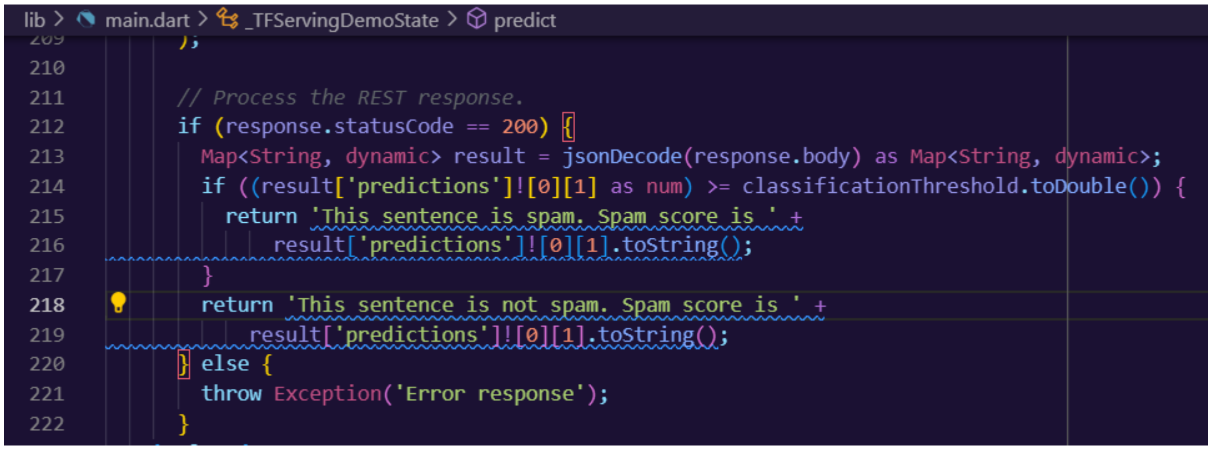
2. Add this code to the REST branch:



```
    if (_connectionMode == ConnectionModeType.rest) {
      //Create the REST request.
      final response = await http.post(
        Uri.parse('http://' +
            server +
            ':' +
            restPort.toString() +
            '/v1/models/' +
            modelName +
            ':predict'),
        body: jsonEncode(<String, List<List<int>>>{
          'instances': [_tokenIndices],
        }),
      );
```

Process the REST response from TensorFlow Serving

- Add this code right after the previous code snippet to handle the REST response:

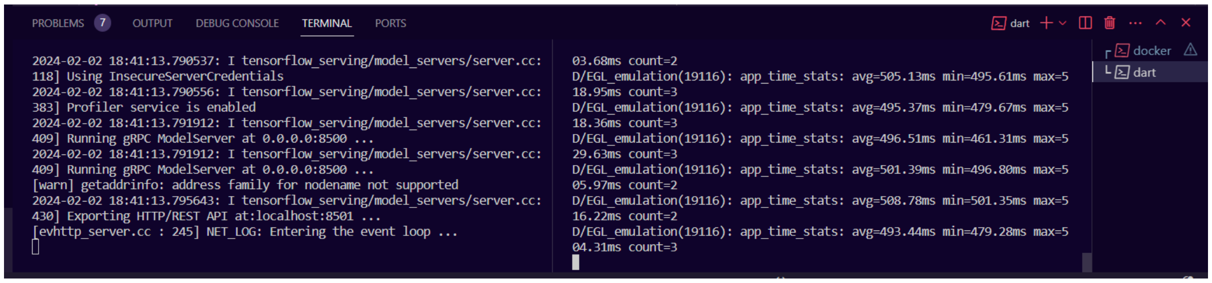


```
      // Process the REST response.
      if (response.statusCode == 200) {
        Map<String, dynamic> result = jsonDecode(response.body) as Map<String, dynamic>;
        if ((result['predictions']![0][1] as num) >= classificationThreshold.toDouble()) {
          return 'This sentence is spam. Spam score is ' +
              result['predictions']![0][1].toString();
        }
        return 'This sentence is not spam. Spam score is ' +
            result['predictions']![0][1].toString();
      } else {
        throw Exception('Error response');
      }
```

The postprocessing code extracts the probability that the input sentence is a spam message from the response and displays the classification result in the UI.

Run it

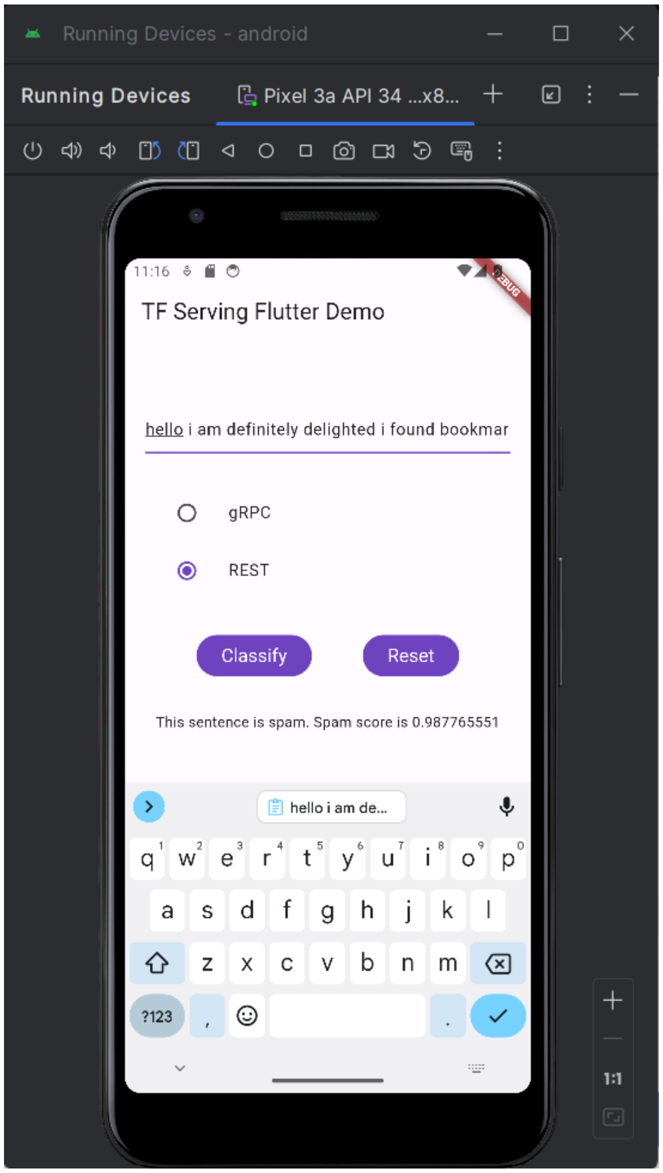
1. Type flutter run in a new terminal. Make sure docker is running on the same time.



```
2024-02-02 18:41:13.790537: I tensorflow_serving/model_servers/server.cc:
118] Using InsecureServerCredentials
2024-02-02 18:41:13.790556: I tensorflow_serving/model_servers/server.cc:
383] Profiler service is enabled
2024-02-02 18:41:13.791912: I tensorflow_serving/model_servers/server.cc:
409] Running gRPC ModelServer at 0.0.0.0:8500 ...
2024-02-02 18:41:13.791912: I tensorflow_serving/model_servers/server.cc:
409] Running gRPC ModelServer at 0.0.0.0:8500 ...
[warn] getaddrinfo: address family for nodename not supported
2024-02-02 18:41:13.795643: I tensorflow_serving/model_servers/server.cc:
430] Exporting HTTP/REST API at:localhost:8501 ...
[evhttp_server.cc : 245] NET_LOG: Entering the event loop ...
```

```
03.68ms count=2
D/EGL_emulation(19116): app_time_stats: avg=505.13ms min=495.61ms max=5
18.95ms count=3
D/EGL_emulation(19116): app_time_stats: avg=495.37ms min=479.67ms max=5
18.36ms count=3
D/EGL_emulation(19116): app_time_stats: avg=496.51ms min=461.31ms max=5
29.63ms count=3
D/EGL_emulation(19116): app_time_stats: avg=501.39ms min=496.80ms max=5
05.97ms count=2
D/EGL_emulation(19116): app_time_stats: avg=508.78ms min=501.35ms max=5
16.22ms count=2
D/EGL_emulation(19116): app_time_stats: avg=493.44ms min=479.28ms max=5
04.31ms count=3
```

2. Enter some text and then select **REST > Classify**.


Running Devices - android
Running Devices
Pixel 3a API 34 ...x8...
11:16
DEBUG
TF Serving Flutter Demo
hello i am definitely delighted i found bookmar
gRPC
REST
Classify
Reset
This sentence is spam. Spam score is 0.987765551
hello i am de...
?123
1:1

Note: Depending on your training procedure and the input sentence, the model may predict a different spam score. For example, "hello i am definitely delighted i found bookmarking site www winbig com very profitable" is an example sentence that generally receives a high spam score.

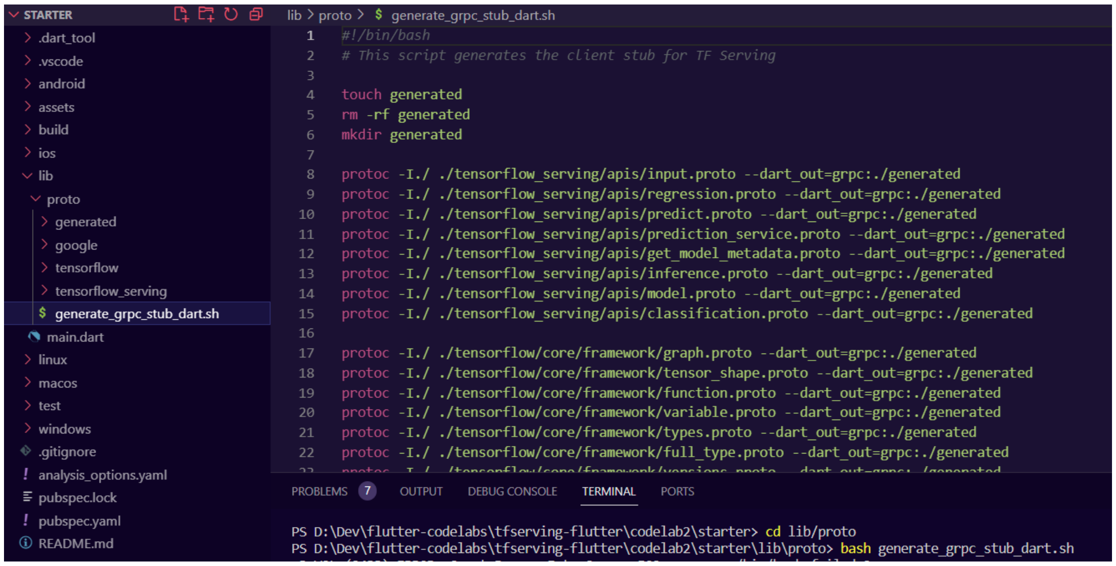
Connect the Flutter app with TensorFlow Serving through gRPC

In addition to REST, TensorFlow Serving also supports gRPC.

gRPC is a modern, open source, high-performance Remote Procedure Call (RPC) framework that can run in any environment. It can efficiently connect services in, and across, data centers with pluggable support for load balancing, tracing, health checking, and authentication. It's been observed that gRPC is more performant than REST in practice.

- In the terminal, navigate to the starter/lib/proto/ folder and generate the stub:

bash generate_grpc_stub_dart.sh

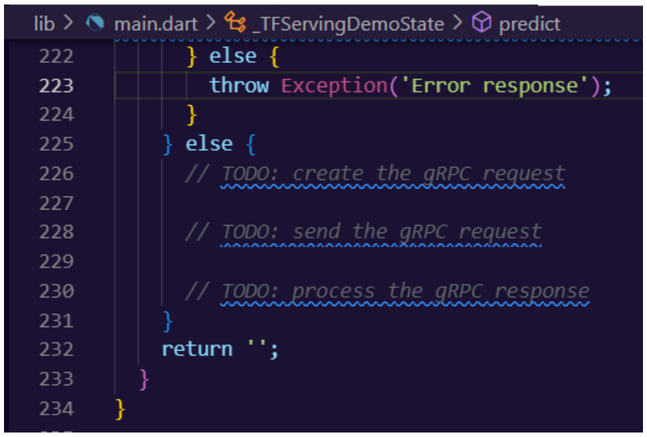


```bash
#!/bin/bash
# This script generates the client stub for TF Serving

touch generated
rm -rf generated
mkdir generated

protoc -I./ ./tensorflow_serving/apis/input.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow_serving/apis/regression.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow_serving/apis/predict.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow_serving/apis/prediction_service.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow_serving/apis/get_model_metadata.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow_serving/apis/inference.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow_serving/apis/model.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow_serving/apis/classification.proto --dart_out=grpc:./generated

protoc -I./ ./tensorflow/core/framework/graph.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow/core/framework/tensor_shape.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow/core/framework/function.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow/core/framework/variable.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow/core/framework/types.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow/core/framework/full_type.proto --dart_out=grpc:./generated
protoc -I./ ./tensorflow/core/framework/versions.proto --dart_out=grpc:./generated
```

```
PS D:\Dev\flutter-codelabs\tfserving-flutter\codelab2\starter> cd lib/proto
PS D:\Dev\flutter-codelabs\tfserving-flutter\codelab2\starter\lib\proto> bash generate_grpc_stub_dart.sh
```

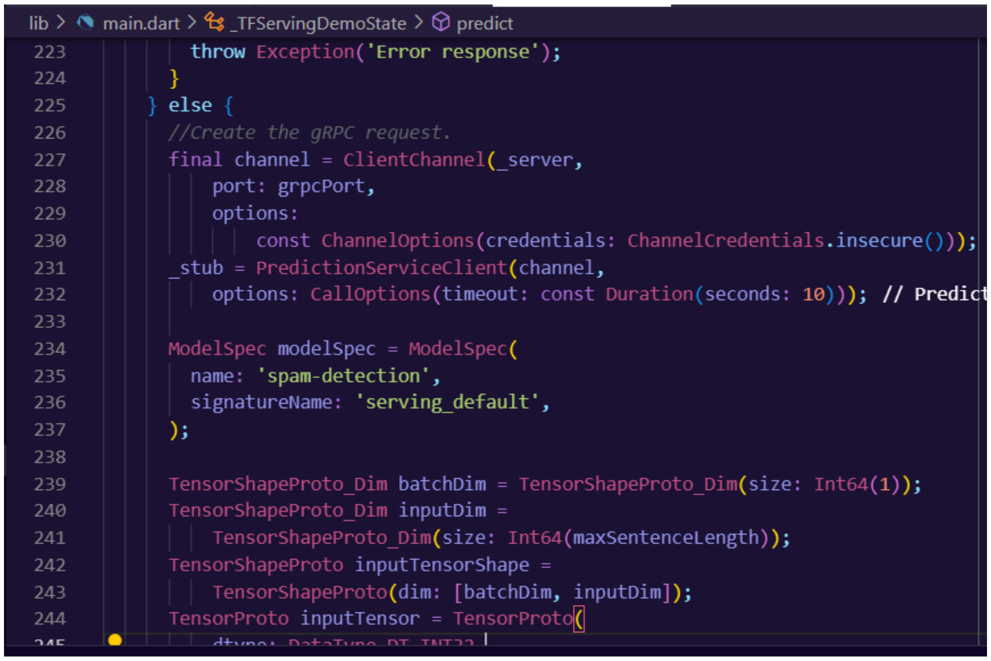
Create the gRPC request

Similar to the REST request, you create the gRPC request in the gRPC branch.



```dart
        } else {
          throw Exception('Error response');
        }
      } else {
        // TODO: create the gRPC request

        // TODO: send the gRPC request

        // TODO: process the gRPC response
      }
      return '';
    }
  }
```

Add this code to create the gRPC request:

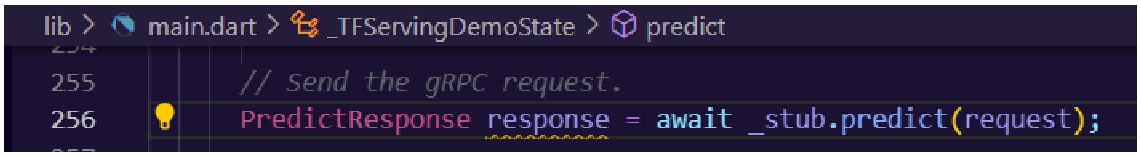


```dart
          throw Exception('Error response');
        }
      } else {
        //Create the gRPC request.
        final channel = ClientChannel(_server,
            port: grpcPort,
            options:
                const ChannelOptions(credentials: ChannelCredentials.insecure()));
        _stub = PredictionServiceClient(channel,
            options: CallOptions(timeout: const Duration(seconds: 10))); // Predict

        ModelSpec modelSpec = ModelSpec(
          name: 'spam-detection',
          signatureName: 'serving_default',
        );

        TensorShapeProto_Dim batchDim = TensorShapeProto_Dim(size: Int64(1));
        TensorShapeProto_Dim inputDim =
            TensorShapeProto_Dim(size: Int64(maxSentenceLength));
        TensorShapeProto inputTensorShape =
            TensorShapeProto(dim: [batchDim, inputDim]);
        TensorProto inputTensor = TensorProto(
```

The input and output tensor names could differ from model to model, even if the model architectures are the same. Make sure to update them if you train your own model.

Send the gRPC request to TensorFlow Serving

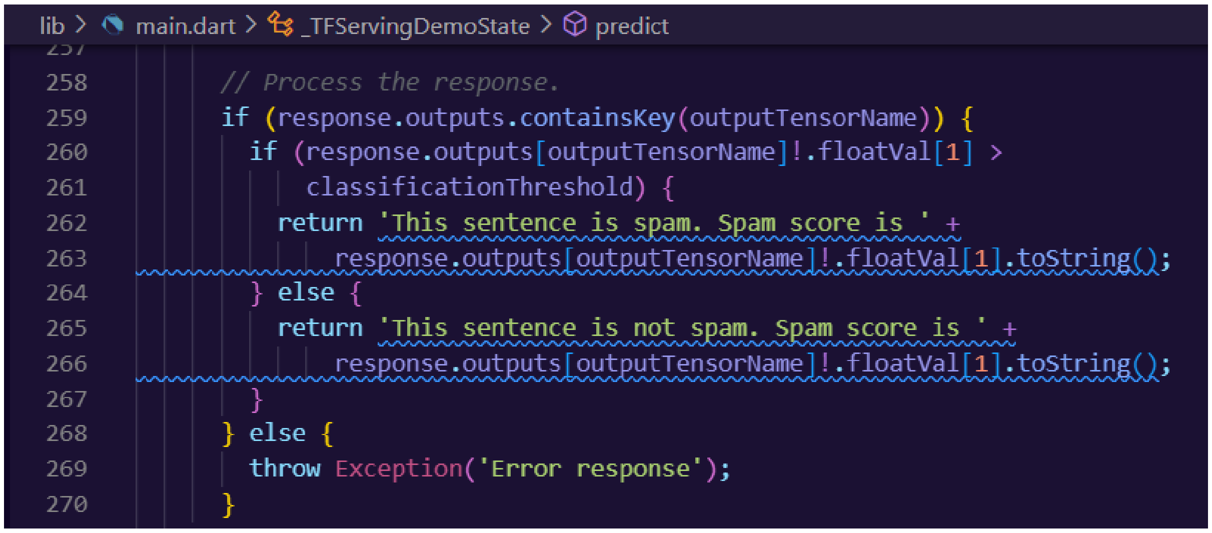
Add this code after the previous code snippet to send the gRPC request to TensorFlow Serving:



```dart
        // Send the gRPC request.
        PredictResponse response = await _stub.predict(request);
```

Process the gRPC response from TensorFlow Serving

Add this code after the previous code snippet to implement the callback functions to handle the response:



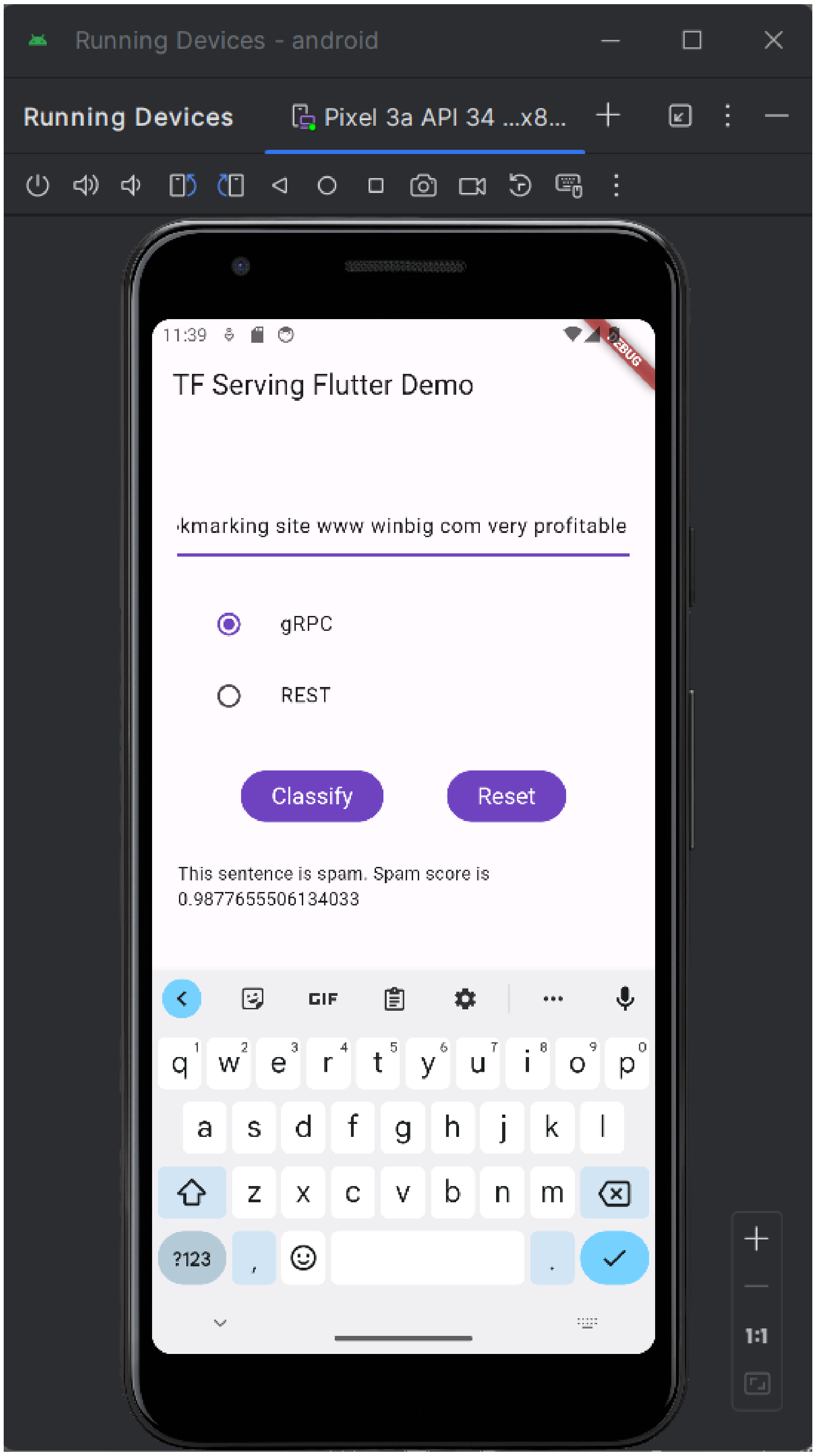
lib > main.dart > _TFServingDemoState > predict

```dart
      // Process the response.
      if (response.outputs.containsKey(outputTensorName)) {
        if (response.outputs[outputTensorName]!.floatVal[1] >
            classificationThreshold) {
          return 'This sentence is spam. Spam score is ' +
              response.outputs[outputTensorName]!.floatVal[1].toString();
        } else {
          return 'This sentence is not spam. Spam score is ' +
              response.outputs[outputTensorName]!.floatVal[1].toString();
        }
      } else {
        throw Exception('Error response');
      }
```

Now the postprocessing code extracts the classification result from the response and displays it in the UI.

Run it

1. Type flutter run in a new terminal. Make sure docker is running on the same time.
2. Enter some text and then select **gRPC > Classify**.



kmarking site www winbig com very profitable

gRPC [✓ gRPC] REST

This sentence is spam. Spam score is 0.9877655506134033