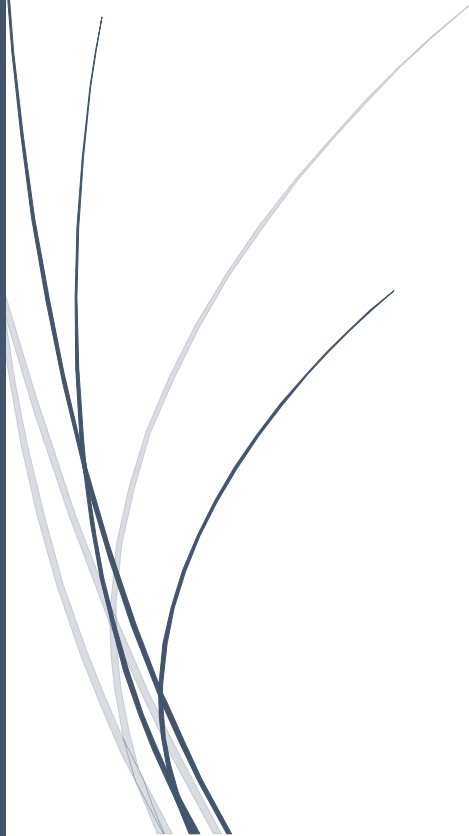




2/3/2024

# Integrating Machine Learning Model in a Flutter application.



Shubham Ramma

BY: KRITIKA BISSESSUR

## Contents

1.1	Brief overview .....	1
1.2	Steps.....	1
1.3	Final Output .....	5
2	Task 2 .....	6
2.1	Brief overview of task .....	6
2.2	Steps.....	6
3	Task 3 .....	9
3.1	Brief Overview.....	9
3.2	Steps.....	9
3.3	Final Output .....	20

## Task-1

### TensorFlow Lite Tutorial for Flutter: Image Classification

<https://www.kodeco.com/37077010-tensorflow-lite-tutorial-for-flutter-image-classification>

Provide a documentation with a short description and screenshots for all the steps of the lab and the Dart files.

## 1.1 Brief overview

In this Flutter guide, I'll create a mobile application named Plant Recognizer, employing machine learning to identify plants from images. The tutorial involves utilizing the Teachable Machine platform, TensorFlow Lite, and the Flutter package called tflite\_flutter.

By the conclusion of this guide, I gained experience in:

- Implementing machine learning in a mobile app.
- Training a model through Teachable Machine.
- Integrating and utilizing TensorFlow Lite with the tflite\_flutter package.
- Developing a mobile app for plant recognition based on images.

## 1.2 Steps

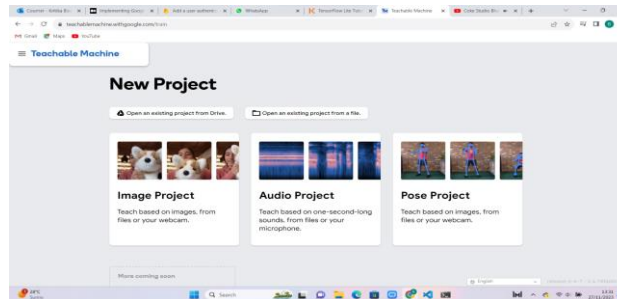
1. Accessing the Training Tool:

Initiate the training process by visiting <https://teachablemachine.withgoogle.com> and clicking on "Get Started."



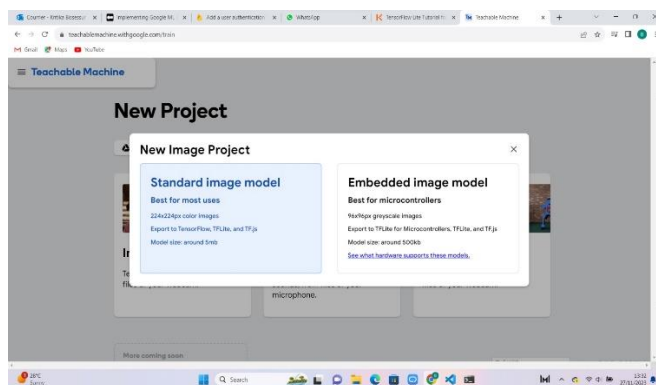
## 2. Selecting Project Type:

When prompted, make the choice to go for an "Image Project" to define the nature of your project.



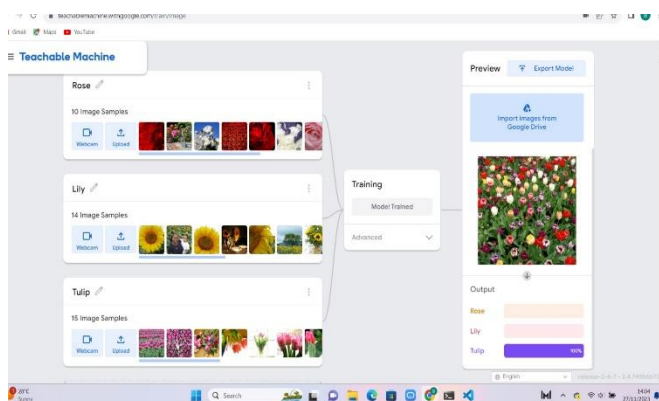
## 3. Choosing Model Type:

Opt for the "Standard Image Model" as you're not creating a model for a microcontroller.



## 4. Class and Label Configuration & Uploading Training Samples:

Within the training tool interface, make the choice to add classes and customize labels for each class based on your project requirements. Then, I populate your training dataset by making the choice to upload samples for each class. Click "Upload" and drag the relevant plant type folder into the designated panel.

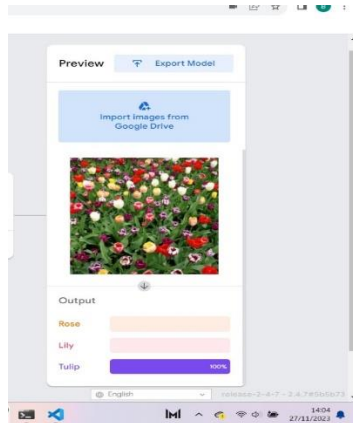


## 5. Initiating Model Training:

Make the choice to start the model training process by clicking "Train Model."

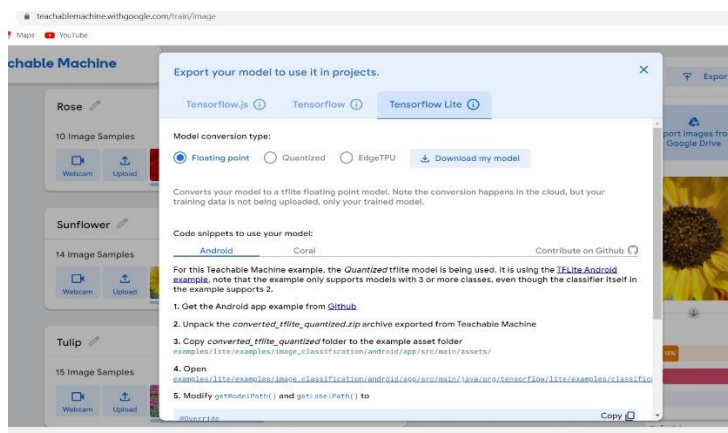
## 6. Testing the Trained Model:

Following the training phase, assess the model's performance by choosing to use images from the samples-test folder.



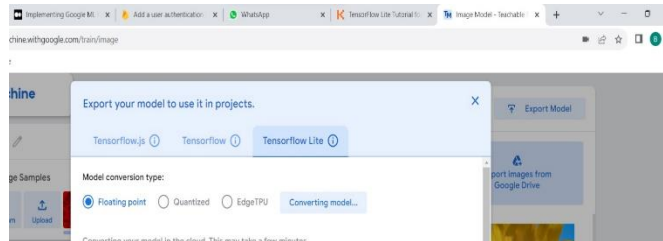
## 7. Exporting the Trained Model:

Extract the trained model by choosing to click "Export Model" on the Preview panel.



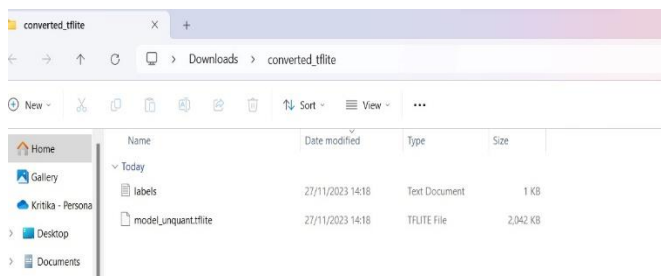
## 8. Selecting TensorFlow Lite for Mobile:

During export, make the choice to opt for TensorFlow Lite to ensure compatibility with mobile platforms.



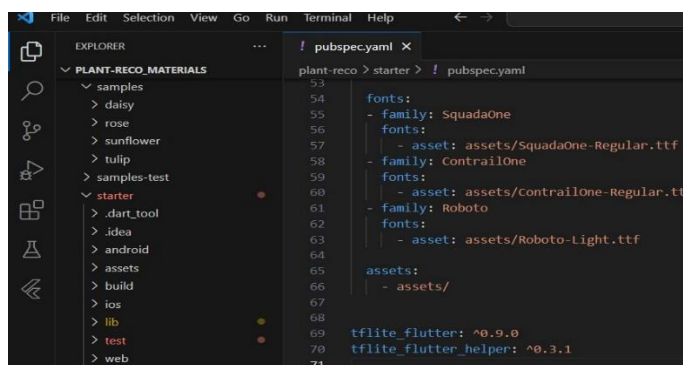
## 9. Downloading and Integrating the Model:

Make the choice to select "Floating point conversion type" for enhanced predictive performance. Click "Download my model" to obtain the converted model file. After receiving the converted\_tflite.zip, decompress it, and transfer labels.txt and model\_unquant.tflite to the ./assets folder in your starter project. The labels.txt file contains class labels, and model\_unquant.tflite is the TensorFlow Lite model designed for mobile use.

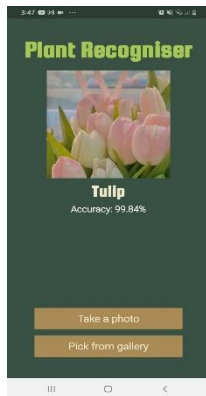


## 10. Get dependencies

I open the starter project in VS Code and run flutter pub dependencies



### 1.3 Final Output



## 2 Task 2

### Task-2

Create an image classification Flutter application using the following
You can choose any fruit or vegetable to classify
Datasets - <a href="https://www.kaggle.com/datasets/moltean/fruits">https://www.kaggle.com/datasets/moltean/fruits</a>
Use the teachable machine web site to train and export your machine learning model TFLite
<a href="https://teachablemachine.withgoogle.com/">https://teachablemachine.withgoogle.com/</a>
Provide a documentation with a short description and screenshots for all the steps of the lab and the Dart files.

### 2.1 Brief overview of task

In this Flutter guide, I created a mobile application named image classification for fruits and vegetables employing machine learning to identify fruits and vegetables from images. The tutorial involves utilizing the Teachable Machine platform, TensorFlow Lite, and the Flutter package called `tflite_flutter`.

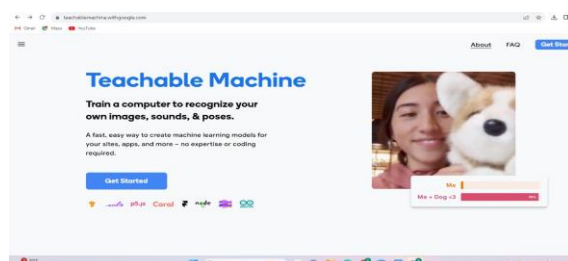
By the conclusion of this guide, I gained expertise in:

- Implementing machine learning in a mobile app.
- Training a model through Teachable Machine.
- Integrating and utilizing TensorFlow Lite with the `tflite_flutter` package.
- Developing a mobile app for plant recognition based on images.

### 2.2 Steps

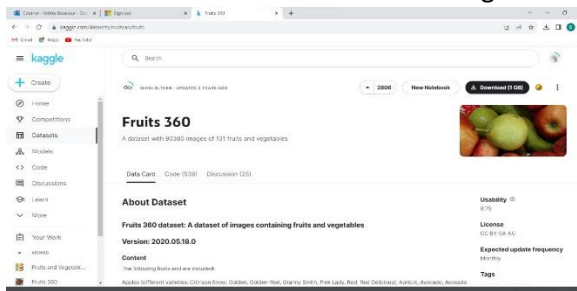
#### 1. Initiating Training Tool Access:

I commence the process by navigating to [Teachable Machine](<https://teachablemachine.withgoogle.com>) and selecting "Get Started."

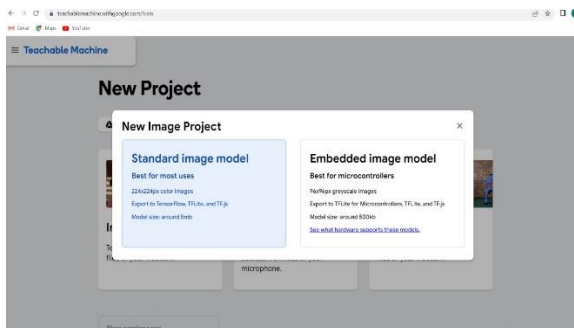




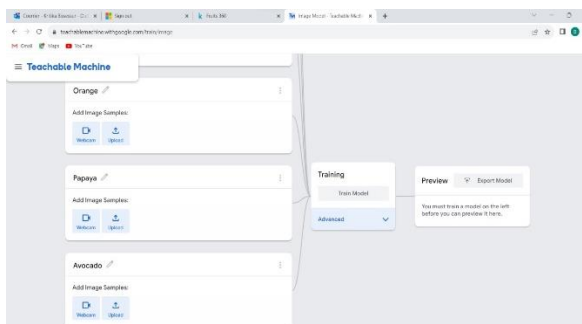
2. Downloading fruit Kaggle set  
I should now download the fruit and vegetable set



3. Model Type Choice:  
I select the "Standard Image Model" as it's not intended for a microcontroller.

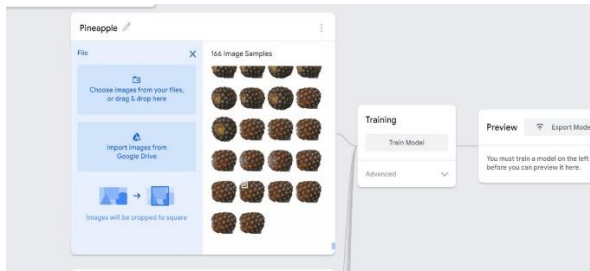


4. Configuring Classes and Labels:  
I add classes and tailor labels to meet the project's requirements. I have added a mixture of fruits and vegetables



## 5. Uploading Training Samples:

I enrich the dataset by uploading samples for each class.



## 6. Commencing Model Training:\*

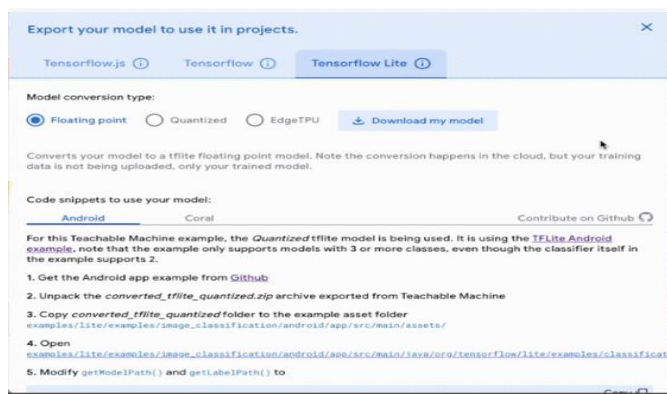
I kick off the model training phase by clicking "Train Model."

## 7. Performance Evaluation:

After training, I assess the model's performance using images from the samples-test folder.

## 8. Exporting the Trained Model:\*

I extract the trained model by selecting "Export Model" on the Preview panel.



## 9. TensorFlow Lite for Mobile:

I ensure mobile compatibility by opting for TensorFlow Lite during the export process.

## 10. Model Download and Integration:

I download the converted model file (converted\_tflite.zip), decompress it, and transfer labels.txt and model\_unquant.tflite to the ./assets folder in my starter project. The labels.txt file contains class labels, and model\_unquant.tflite is the TensorFlow Lite model tailored for mobile use.

### 3 Task 3

#### Task-3

Create a Flutter app to classify texts

<https://developers.google.com/codelabs/classify-texts-flutter-tensorflow-serving#0>

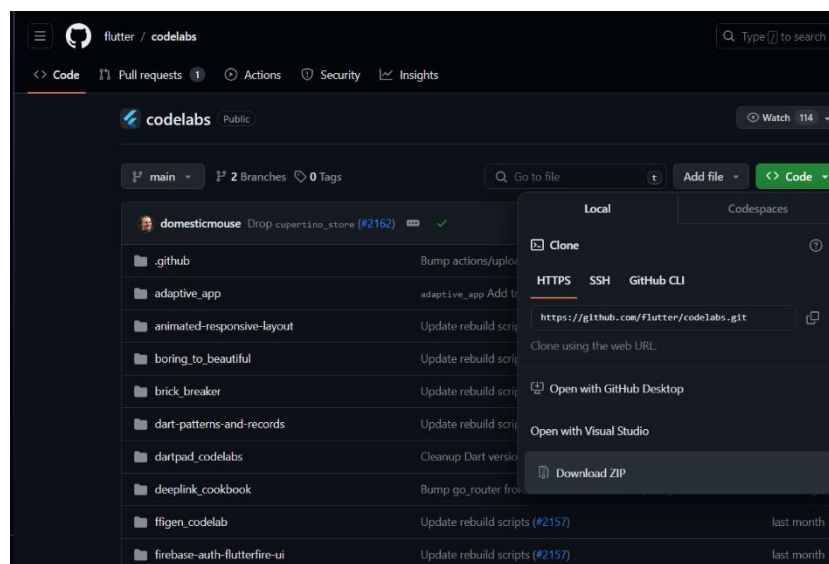
Provide a documentation with a short description and screenshots for all the steps of the lab and the Dart files.

#### 3.1 Brief Overview

The tutorial covers two communication protocols—REST and gRPC. We are instructed to download the code from a specific GitHub repository, where we find the necessary resources. The key tasks include setting up the Flutter app, making requests to TensorFlow Serving using both REST and gRPC, and handling text classification inferences.

#### 3.2 Steps

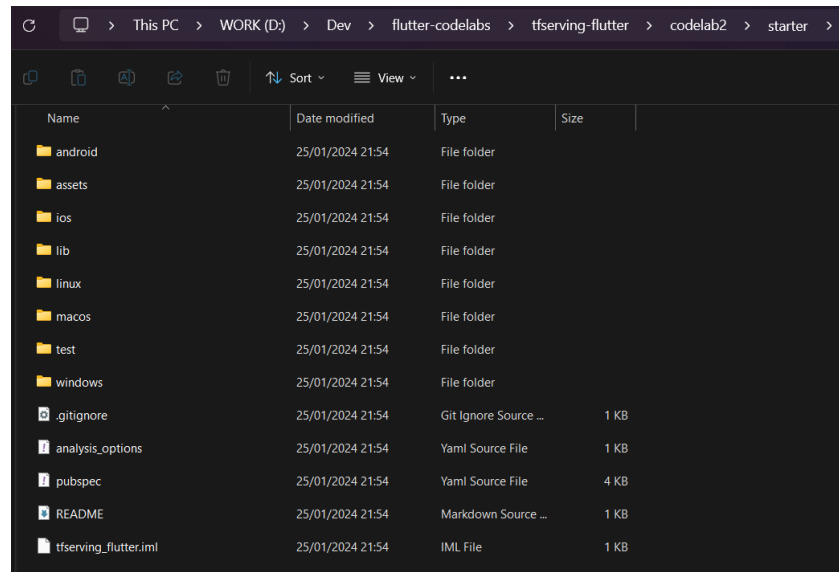
1. To access the code for this codelab, I should visit the GitHub repository and follow these steps: Click on the "Code" button and choose "Download zip" from the dropdown. This will download a zip file containing all the required code. Extract the contents after downloading



2. Now, I should unzip the downloaded zip file to unpack a codelabs-main root folder with all the resources that I need.

For this codelab, I only need the files in the `tf-serving-flutter/codelab2` subdirectory in the repository, which contains two folders:

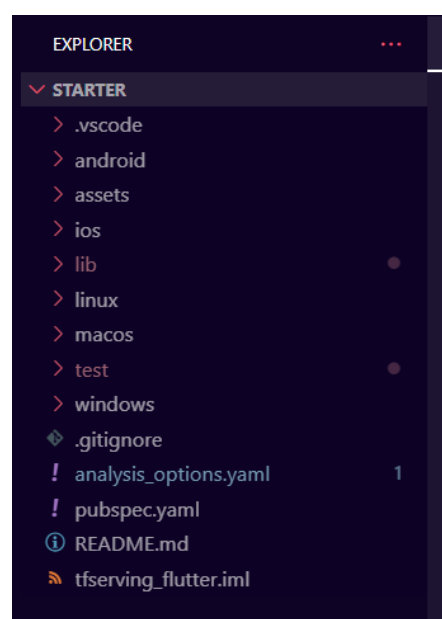
- The starter folder contains the starter code that you build upon for this codelab.



### 3. Download the dependencies for the project

In Visual Studio Code (VS Code), I begin by selecting "File" and then choosing "Open folder." Navigate to the source code folder I downloaded earlier, specifically the "starter" folder. Once selected, if a dialog appears prompting me to download the necessary packages for the starter app, click on "Get packages" to initiate the download.

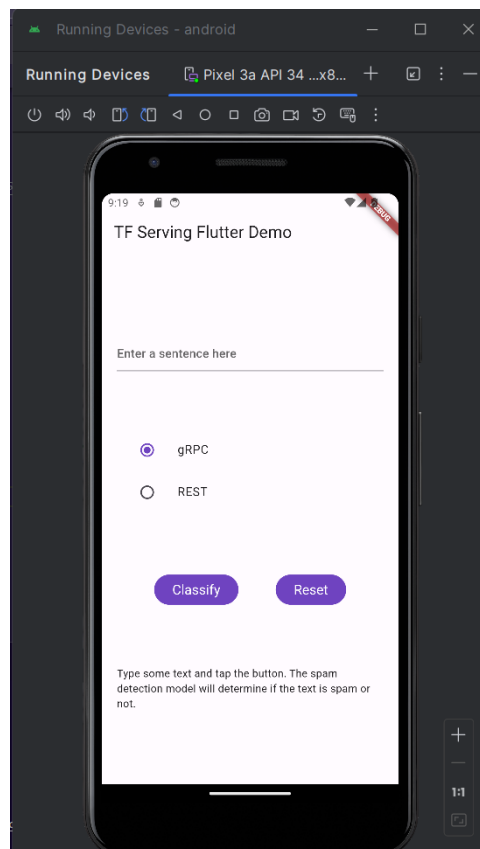
In case I don't encounter this dialog, I manually open MY terminal within the "starter" folder and execute the command `flutter pub get`. This command is crucial for fetching and installing the required Flutter packages essential for the proper functioning of the starter app. This ensures that my development environment is set up with all the dependencies needed to proceed with the codelab.



```
PS D:\Dev\flutter-codelabs\tfserving-flutter\codelab2\starter> flutter pub get
Resolving dependencies... (2.2s)
+ archive 3.4.10
+ args 2.4.2
+ async 2.11.0
+ boolean_selector 2.1.1
+ characters 1.3.0
+ clock 1.1.1
+ collection 1.18.0
```

#### 4. Run and explore the app

The app should launch on my Android Emulator or iOS Simulator. The UI is straightforward. There's a text field that lets the user type in the text. The user can choose whether to send the data to the backend with REST or gRPC. The backend uses a TensorFlow model to perform text classification on the preprocessed input and returns the classification result to the client app, which updates the UI in turn.



#### 5. Deploy a text-classification model with TensorFlow Serving

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

#### 6. Start TensorFlow Serving

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

```
PS D:\Dev\flutter-codelabs\tfserving-flutter\codelab2\starter> docker pull tensorflow/serving
Using default tag: latest
latest: Pulling from tensorflow/serving
96d54c3075c9: Downloading [>] 277.8kB/27.51MB
ce077e3fadc4: Downloading [=====] 1.996MB/2.65MB
806c774cb78b: Downloading [>] 1.602MB/115.3MB
c588a3276cac: Waiting
050d4101433f: Waiting
[ ]
```

```
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:fdc296313fa4454173c5728fceda38f5d18cdb44c71a9f279ce61bc5818335e
```

```
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\codelab2\starter\m_spam_saved_model\models\spam-detection" tensorflow/serving
2024-02-02 18:41:13.470561: I external/org_tensorflow/tensorflow/core/util/port.cc:111] OneDNN custom operations are on. You may see slightly different numerical results 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_path: /models/spam-detection
2024-02-02 18:41:13.473280: 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:196] (re-)loading model: spam-detection
2024-02-02 18:41:13.630192: I tensorflow_serving/core/basic_manager.cc:790] 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 18:41:13.630288: 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-detection/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 instructions 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/reader.cc:231] Restoring SavedModel bundle.
2024-02-02 18:41:13.734929: I external/org_tensorflow/tensorflow/cc/saved_model/reader.cc:217] Running initialization op on SavedModel bundle at path: /models/spam-detection/123
2024-02-02 18:41:13.740667: I external/org_tensorflow/tensorflow/cc/saved_model/reader.cc:316] SavedModel load for tags { serve }; status: success: OK. Took 107852 microseconds.
2024-02-02 18:41:13.741774: I tensorflow_serving/servables/tensorflow/saved_model_warmp_util.cc:80] No warmup data file found at /models/spam-detection/123/assets.warmup
[ ]
```

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:

```
PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS

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_warmp
extra/tf_serving_warmp_requests
2024-02-02 18:41:13.788668: I tensorflow_serving/core/loader_harness.cc:95] Successfu
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 ...
[ ]
```

```
localhost:8501/v1/models/spam-detection/metadata
{
  "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"
          }
        }
      }
    }
  }
}
```

## 7. Tokenize input sentence

Before sending client requests to TensorFlow Serving, it's crucial to tokenize the input sentences. The model expects a list of 20 integer numbers instead of raw strings. Tokenization involves mapping individual words to integers based on a vocabulary dictionary. For instance, the sentence "buy book online to learn more" might be tokenized as [32, 79, 183, 10, 224, 631, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]. The specific numbers depend on the vocabulary dictionary. This process ensures proper input format for the backend's classification.

- In the lib/main.dart file, I add this code to the predict() method to build the \_vocabMap vocabulary dictionary.

```
lib > main.dart > _TFServingDemoState > predict
164 // Build _vocabMap if empty.
165 if (_vocabMap.isEmpty) {
166   final vocabFileString = await rootBundle.loadString(vocabFile);
167   final lines = vocabFileString.split('\n');
168   for (final l in lines) {
169     if (l != "") {
170       var wordAndIndex = l.split(' ');
171       (_vocabMap)[wordAndIndex[0]] = int.parse(wordAndIndex[1]);
172     }
173   }
174 }
```

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

```
lib > main.dart > _TF-servingDemoState > predict
173
174 }
175
176 // Tokenize the input sentence.
177 final inputWords = _inputSentenceController.text
178   .toLowerCase()
179   .replaceAll(RegExp('[^a-z ]'), '')
180   .split(' ');
181 // Initialize with padding token.
182 _tokenIndices = List.filled(maxSentenceLength, 0);
183 var i = 0;
184 for (final w in inputWords) {
185   if ((_vocabMap).containsKey(w)) {
186     _tokenIndices[i] = (_vocabMap)[w]!;
187     i++;
188   }
189
190 // Truncate the string if longer than maxSentenceLength.
191 if (i >= maxSentenceLength - 1) {
192   break;
193 }
194 }
```

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

#### 8. 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:

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



## 9. Create and send the REST request to TensorFlow Serving

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

```
lib > main.dart > _TFServingDemoState > predict
195
196   if (_connectionMode == ConnectionModeType.rest) {
197       // TODO: create and send the REST request
198
199       // TODO: process the REST response
200   } else {
201       // TODO: create the gRPC request
202
203       // TODO: send the gRPC request
204
205       // TODO: process the gRPC response
206   }
207   return '';
208 }
```

## 2. Then this code to the REST branch:

```
lib > main.dart > _TFServingDemoState > predict
195
196   if (_connectionMode == ConnectionModeType.rest) {
197       //Create the REST request.
198       final response = await http.post(
199         Uri.parse('http://' +
200           server +
201           ':' +
202           restPort.toString() +
203           '/v1/models/' +
204           modelName +
205           ':predict'),
206         body: jsonEncode(<String, List<List<int>>>{
207           'instances': [_tokenIndices],
208         }),
209       );
```


## Process the REST response from TensorFlow Serving

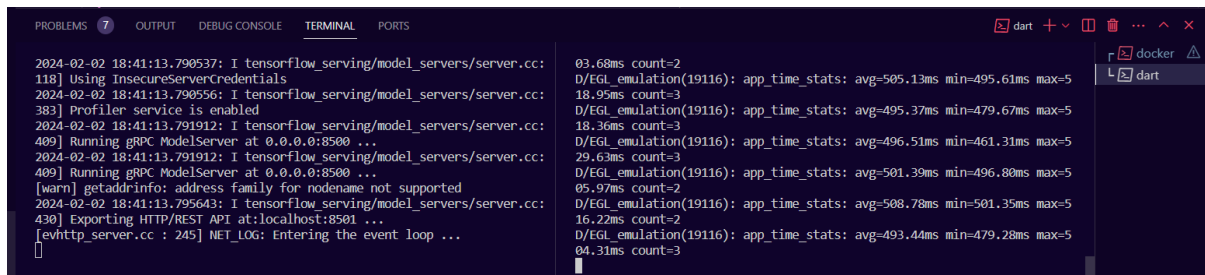
- Add this code right after the previous code snippet to handle the REST response:

```
lib > main.dart > _TFServingDemoState > predict
209
210
211 // Process the REST response.
212 if (response.statusCode == 200) {
213   Map<String, dynamic> result = jsonDecode(response.body) as Map<String, dynamic>;
214   if ((result['predictions']![0][1] as num) >= classificationThreshold.toDouble()) {
215     return 'This sentence is spam. Spam score is ' +
216         result['predictions']![0][1].toString();
217   }
218   return 'This sentence is not spam. Spam score is ' +
219       result['predictions']![0][1].toString();
220 } else {
221   throw Exception('Error response');
222 }
```

10. 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.

11. Then i will run the app

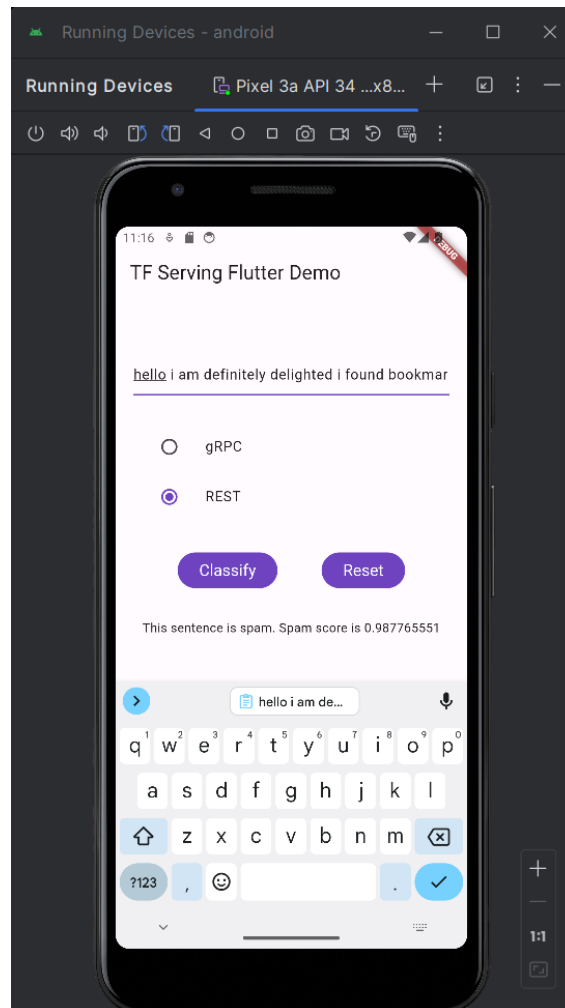
- I click  **Start debugging** and then wait for the app to load.



```
PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS
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=518.95ms count=3
D/EGL_emulation(19116): app_time_stats: avg=495.37ms min=479.67ms max=518.36ms count=3
D/EGL_emulation(19116): app_time_stats: avg=496.51ms min=461.31ms max=529.63ms count=3
D/EGL_emulation(19116): app_time_stats: avg=501.39ms min=496.80ms max=505.97ms count=2
D/EGL_emulation(19116): app_time_stats: avg=508.78ms min=501.35ms max=516.22ms count=2
D/EGL_emulation(19116): app_time_stats: avg=493.44ms min=479.28ms max=504.31ms count=3
[+] docker
[+] dart
```

**Make sure docker is running on the same time.**

- I then enter some text and then select **REST > Classify**.



#### 11. 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.

- ```
bash generate_grpc_stub_dart.sh
```

```
STARTER lib > proto > $ generate_grpc_stub_dart.sh  
1 #!/bin/bash  
2 This script generates the client stub for TF Serving  
3  
4 touch generated  
5 rm -rf generated  
6 mkdir generated  
7  
8 protoc -I./ ./tensorflow_serving/apis/input.proto --dart_out-grpc://generated  
9 protoc -I./ ./tensorflow_serving/apis/regression.proto --dart_out-grpc://generated  
10 protoc -I./ ./tensorflow_serving/apis/predict.proto --dart_out-grpc://generated  
11 protoc -I./ ./tensorflow_serving/apis/prediction_service.proto --dart_out-grpc://generated  
12 protoc -I./ ./tensorflow_serving/apis/get_model_metadata.proto --dart_out-grpc://generated  
13 protoc -I./ ./tensorflow_serving/apis/inference.proto --dart_out-grpc://generated  
14 protoc -I./ ./tensorflow_serving/apis/model.proto --dart_out-grpc://generated  
15 protoc -I./ ./tensorflow_serving/apis/classification.proto --dart_out-grpc://generated  
16  
17 protoc -I./ ./tensorflow/core/framework/graph.proto --dart_out-grpc://generated  
18 protoc -I./ ./tensorflow/core/framework/tensor_shape.proto --dart_out-grpc://generated  
19 protoc -I./ ./tensorflow/core/framework/function.proto --dart_out-grpc://generated  
20 protoc -I./ ./tensorflow/core/framework/variable.proto --dart_out-grpc://generated  
21 protoc -I./ ./tensorflow/core/framework/types.proto --dart_out-grpc://generated  
22 protoc -I./ ./tensorflow/core/framework/full_type.proto --dart_out-grpc://generated  
23 protoc -I./ ./tensorflow/core/framework/op_def.proto --dart_out-grpc://generated  
  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
  
PS D:\Dev\Flutter-code-labs\tf-serving-flutter-flutter(code-lab2)> cd lib/proto  
PS D:\Dev\Flutter-code-labs\tf-serving-flutter-flutter(code-lab2)> starter\lib\proto> bash generate_grpc_stub_dart.sh
```

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

```
lib > main.dart > _TF-serving-demo-state > predict
222     } else {
223         throw Exception('Error response');
224     }
225 } else {
226     // TODO: create the gRPC request
227
228     // TODO: send the gRPC request
229
230     // TODO: process the gRPC response
231 }
232 return '';
233 }
234 }
```

- Add this code to create the gRPC request:

```
lib > main.dart > _TFServingDemoState > predict
223     throw Exception('Error response');
224   }
225   } else {
226     //Create the gRPC request.
227     final channel = ClientChannel(_server,
228       port: grpcPort,
229       options:
230         const ChannelOptions(credentials: ChannelCredentials.insecure()));
231     _stub = PredictionServiceClient(channel,
232       options: CallOptions(timeout: const Duration(seconds: 10))); // Predict
233
234     ModelSpec modelSpec = ModelSpec(
235       name: 'spam-detection',
236       signatureName: 'serving_default',
237     );
238
239     TensorShapeProto_Dim batchDim = TensorShapeProto_Dim(size: Int64(1));
240     TensorShapeProto_Dim inputDim =
241       TensorShapeProto_Dim(size: Int64(maxSentenceLength));
242     TensorShapeProto inputTensorShape =
243       TensorShapeProto(dim: [batchDim, inputDim]);
244     TensorProto inputTensor = TensorProto(
245       dtypes: DataTypes.INT32,
```

**Note:** 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:

```
lib > main.dart > _TFServingDemoState > predict
255     // Send the gRPC request.
256     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
257
258     // Process the response.
259     if (response.outputs.containsKey(outputTensorName)) {
260       if (response.outputs[outputTensorName]!.floatVal[1] >
261         classificationThreshold) {
262         return 'This sentence is spam. Spam score is ' +
263           response.outputs[outputTensorName]!.floatVal[1].toString();
264       } else {
265         return 'This sentence is not spam. Spam score is ' +
266           response.outputs[outputTensorName]!.floatVal[1].toString();
267       }
268     } else {
269       throw Exception('Error response');
270     }
```

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

Run it

1. Click  **Start debugging** and then wait for the app to load.
2. Enter some text and then select **gRPC > Classify**.

### 3.3 Final Output

