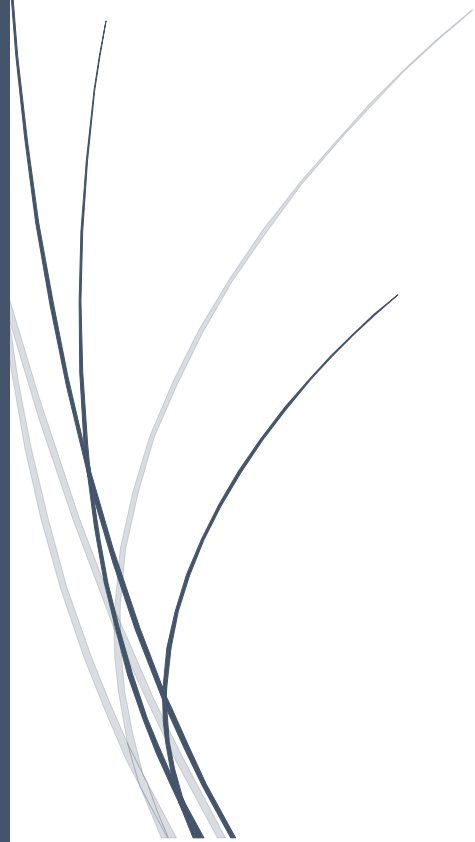
UNIVERSITÉ DES MASCAREIGNES

SAVOIR, C'EST POUVOIR

Université de Limoges

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

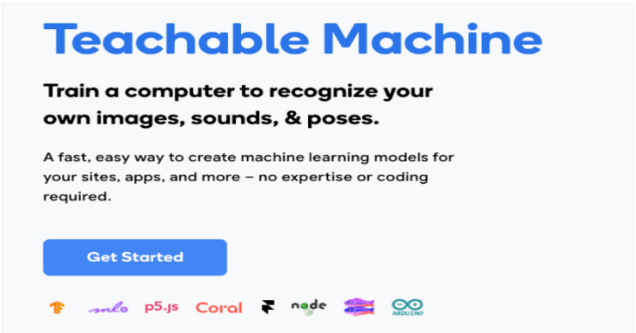
In this Flutter guide, I'll create a mobile application named Plant Recognizer, employing machine learning to identify plants from images. The tutorial involves utilizing the Teachable Machine platform, TensorFlow Lite, and the Flutter package called tflite_flutter.

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

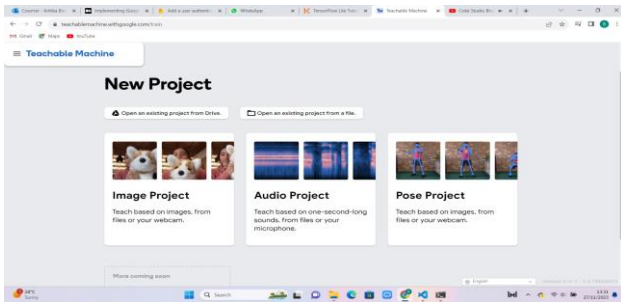
- Implementing machine learning in a mobile app.
- Training a model through Teachable Machine.
- Integrating and utilizing TensorFlow Lite with the tflite_flutter package.
- Developing a mobile app for plant recognition based on images.

1.2 Steps

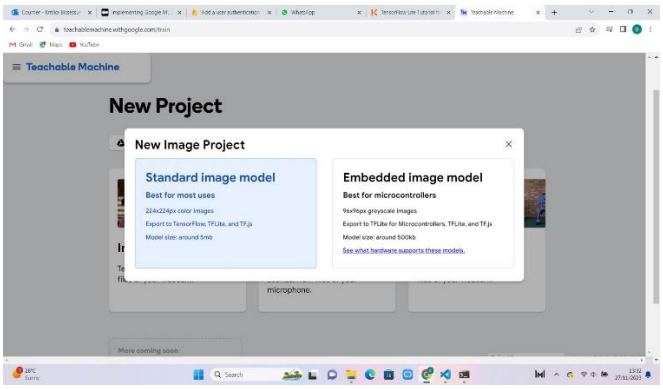
1. Accessing the Training Tool:
   Initiate the training process by visiting https://teachablemachine.withgoogle.com and clicking on "Get Started."



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

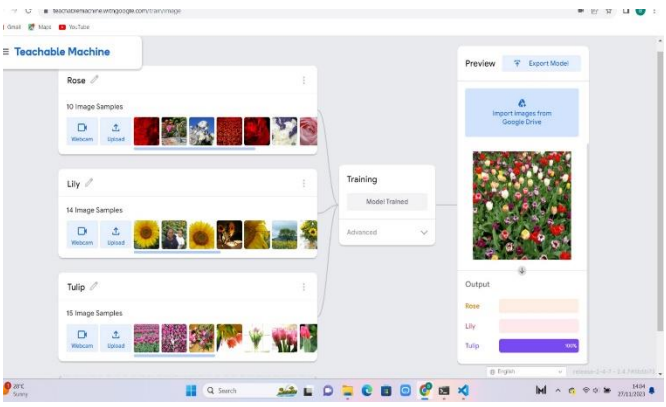


3. Choosing Model Type:
   Opt for the "Standard Image Model" as you're not creating a model for a microcontroller.



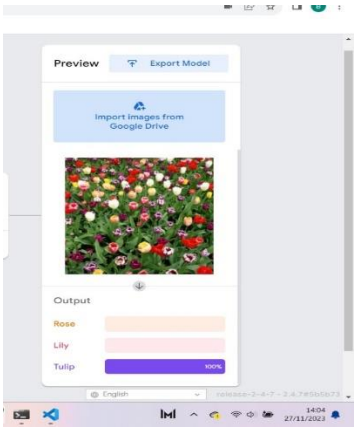
4. Class and Label Configuration & Uploading Training Samples:

   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.

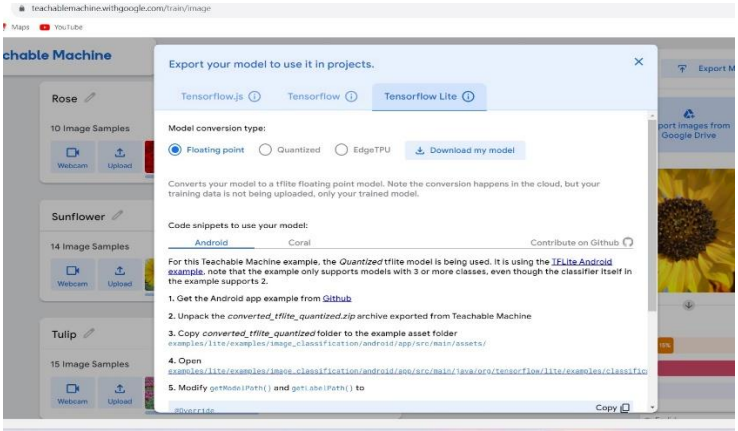


5. Initiating Model Training:
   Make the choice to start the model training process by clicking "Train Model."

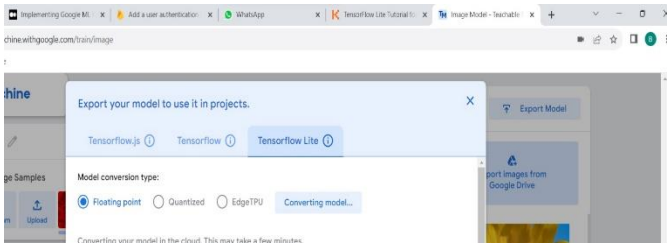
6. Testing the Trained Model:
   Following the training phase, assess the model's performance by choosing to use images from the samples-test folder.



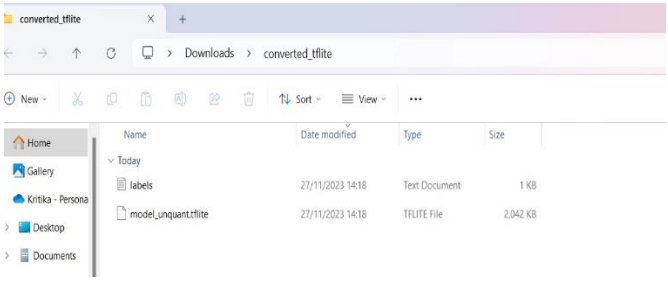
7. Exporting the Trained Model:
   Extract the trained model by choosing to click "Export Model" on the Preview panel.



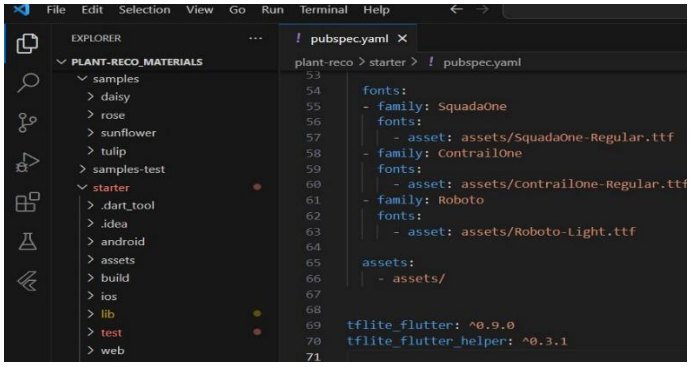
8. Selecting TensorFlow Lite for Mobile:
   During export, make the choice to opt for TensorFlow Lite to ensure compatibility with mobile platforms.



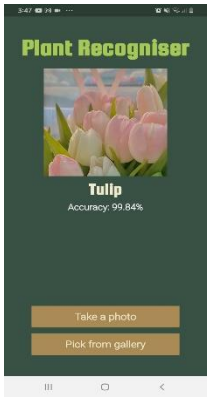
9. Downloading and Integrating the Model:
   Make the choice to select "Floating point conversion type" for enhanced predictive performance. Click "Download my model" to obtain the converted model file. After receiving the converted_tflite.zip, decompress it, and transfer labels.txt and model_unquant.tflite 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 - https://www.kaggle.com/datasets/moltean/fruits
Use the teachable machine web site to train and export your machine learning model TFlite
https://teachablemachine.withgoogle.com/
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

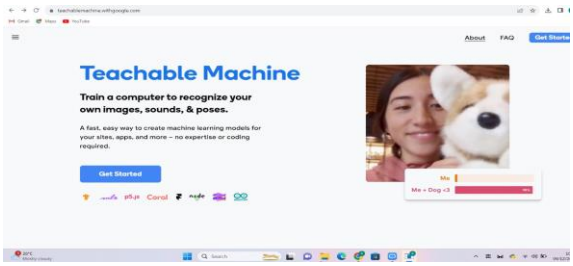
In this Flutter guide, I created a mobile application named image classification for fruits and vegetables employing machine learning to identify fruits and vegetables from images. The tutorial involves utilizing the Teachable Machine platform, TensorFlow Lite, and the Flutter package called tflite_flutter.

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

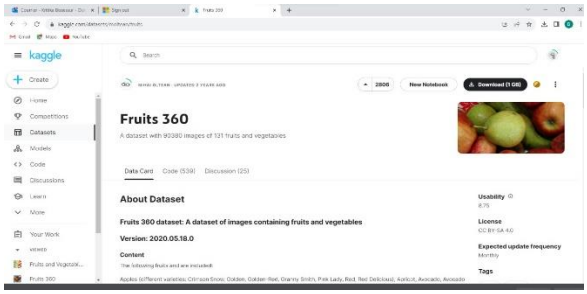
- Implementing machine learning in a mobile app.
- Training a model through Teachable Machine.
- Integrating and utilizing TensorFlow Lite with the tflite_flutter package.
- Developing a mobile app for plant recognition based on images.

2.2 Steps

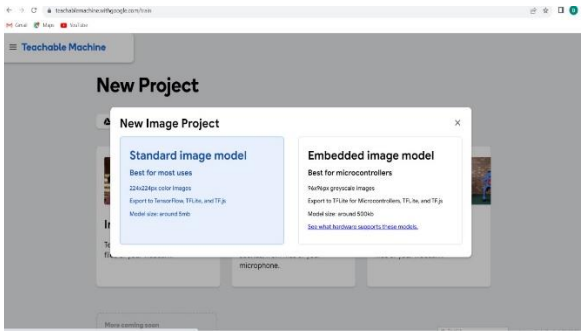
1. Initiating Training Tool Access:
   I commence the process by navigating to [Teachable Machine](https://teachablemachine.withgoogle.com) and selecting "Get Started."



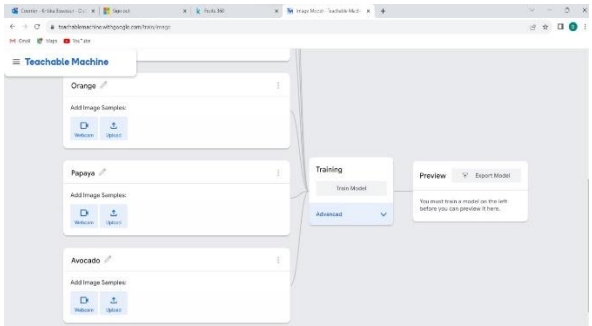
2. Dowloading fruit Kaggle set
   I should now download the fruit and vegetable set



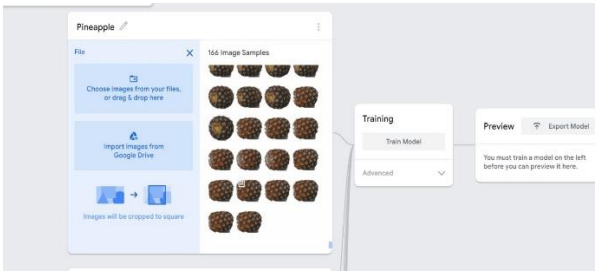
3. Model Type Choice:
   I select the "Standard Image Model" as it's not intended for a microcontroller.



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



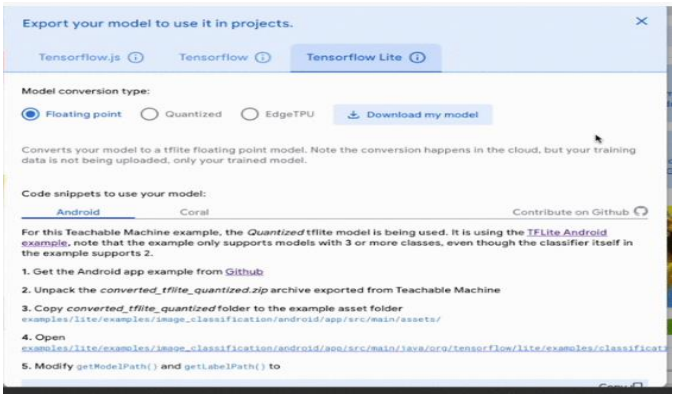
5. Uploading Training Samples:
   I enrich the dataset by uploading samples for each class.



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

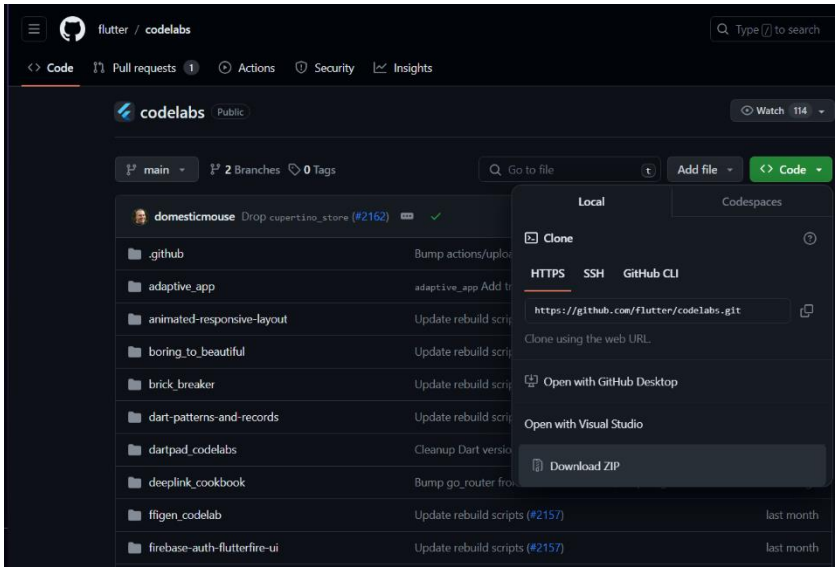
Create a Flutter app to classify texts
https://developers.google.com/codelabs/classify-texts-flutter-tensorflow-serving#0
Provide a documentation with a short description and screenshots for all the steps of the lab and the Dart files.

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 wefind 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

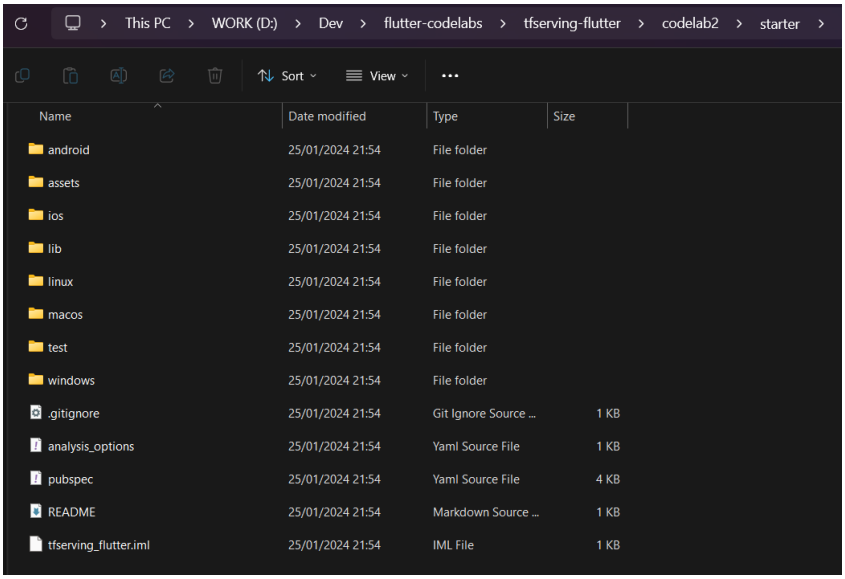
1. To access the code for this codelab, I should visit the GitHub repository and follow these steps: Click on the "Code" button and choose "Download zip" from the dropdown. This will download a zip file containing all the required code. Extract the contents after downloading



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 tfserving-flutter/codelab2 subdirectory in the repository, which contains two folders:

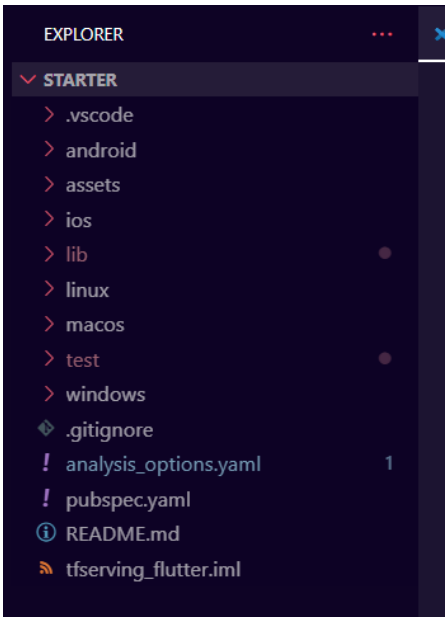
- The starter folder contains the starter code that you build upon for this codelab.



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.

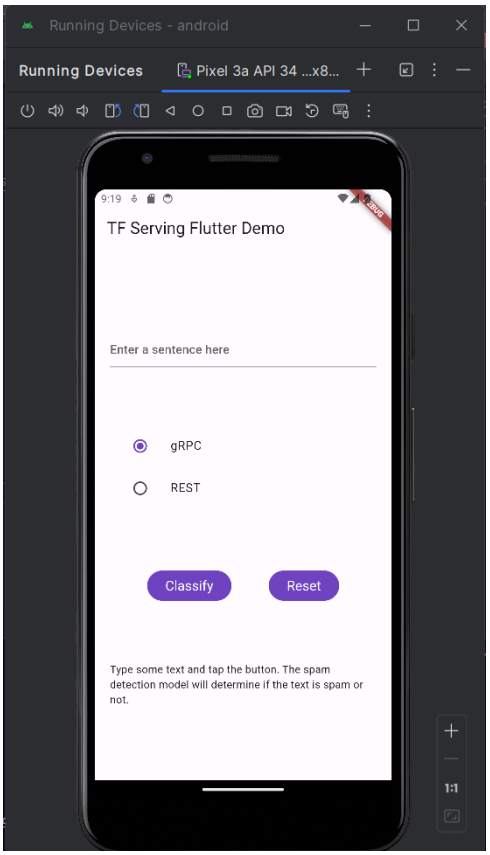
In case I don't encounter this dialog, I manually open MY terminal within the "starter" folder and execute the command `flutter pub get`. This command is crucial for fetching and installing the required Flutter packages essential for the proper functioning of the starter app. This ensures that my development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

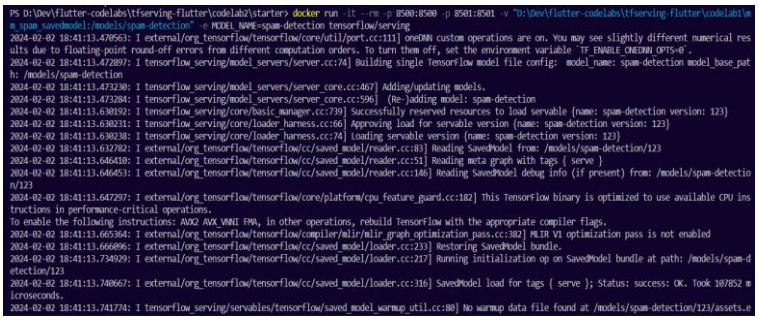
Note: A pretrained SavedModel along with the vocabulary and label files is provided in the tfserving-flutter/codelab1/mm_spam_savedmodel folder.

6. Start TensorFlow Serving

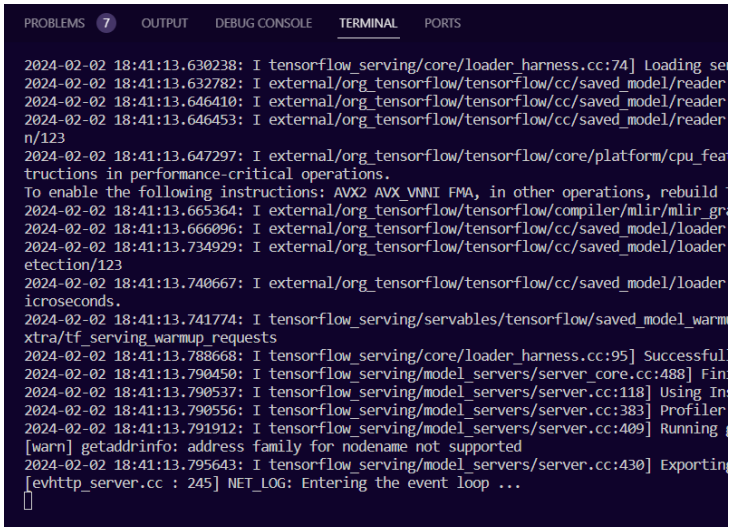
- In my terminal, I start TensorFlow Serving with Docker, but replace the *PATH/TO/SAVEDMODEL* placeholder with the absolute path of the mm_spam_savedmodel folder on mycomputer.

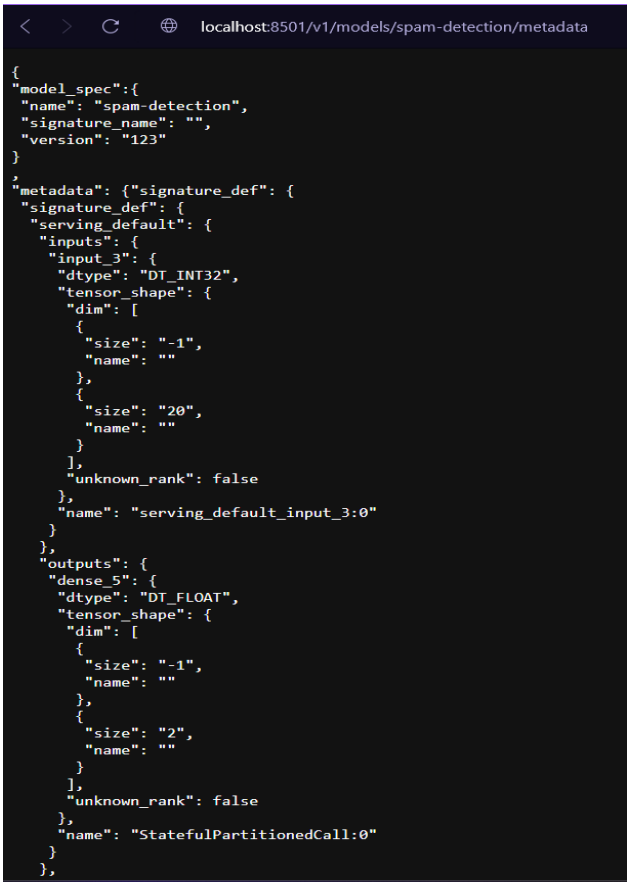
```
PS D:\Dev\flutter-codelabs\tfserving-flutter\codelab2\starter> docker pull tensorflow/serving
Using default tag: latest
latest: Pulling from tensorflow/serving
96d54c3075c9: Downloading [>                                                  ]  277.8kB/27.51MB
ce077e3fadc4: Downloading [==================================>                ]  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:fdc296e313fa4454173c5728fceda38f5d18cdb44c71a9f279ce61bc5818335e
```



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:

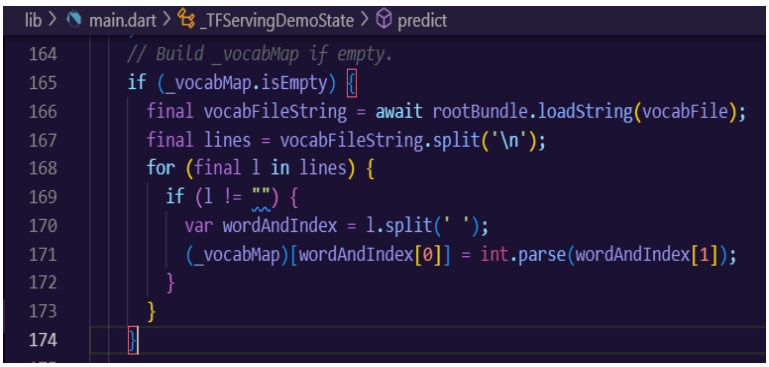




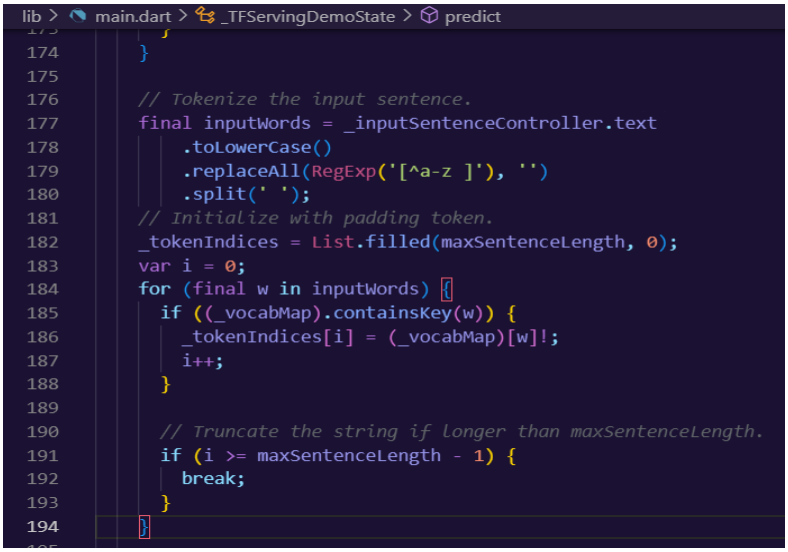
7. Tokenize input sentence

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, 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.



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



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

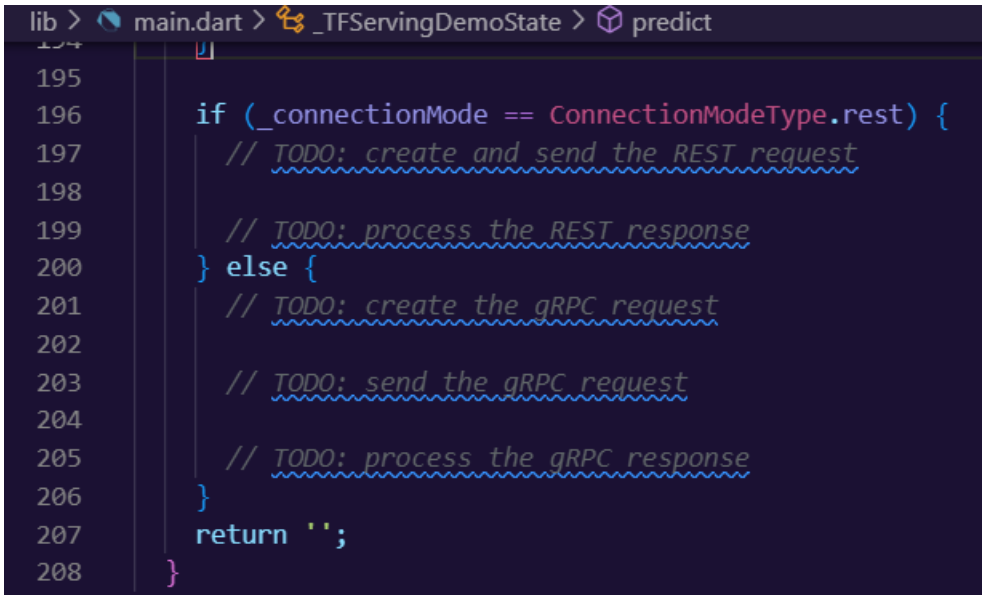
Send requests and receive responses through REST

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

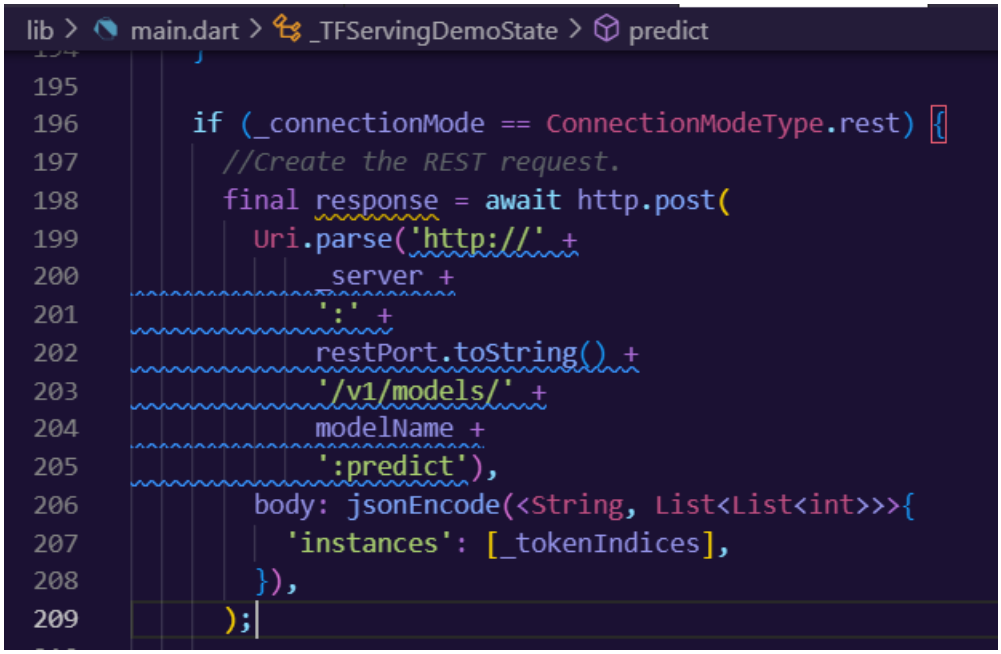
- Create the REST request.
- Send the REST request to TensorFlow Serving.
- Extract the predicted result from the REST response and render the UI.

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:



2. Then this code to the REST branch:



Process the REST response from TensorFlow Serving

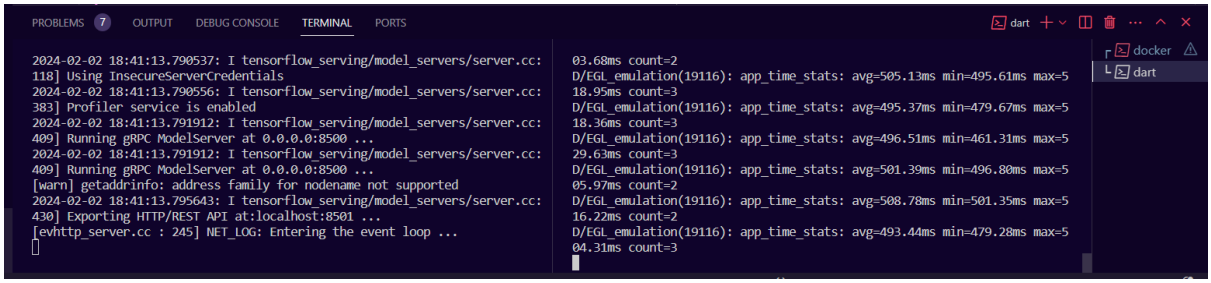
- Add this code right after the previous code snippet to handle the REST response:

```
lib > main.dart > _TFServingDemoState > predict
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.

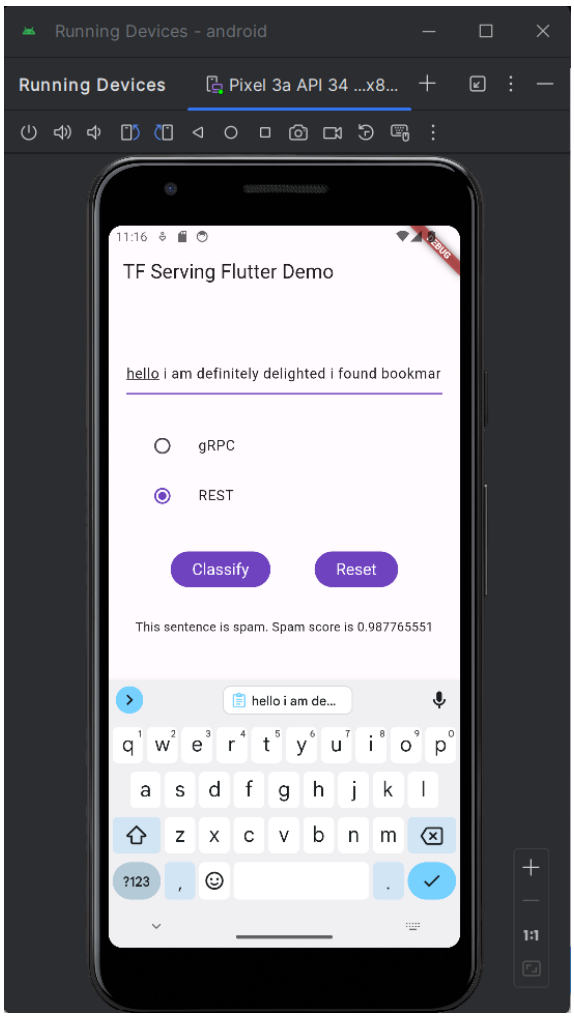
11. Them i will run the app

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



Make sure docker is running on the same time.

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



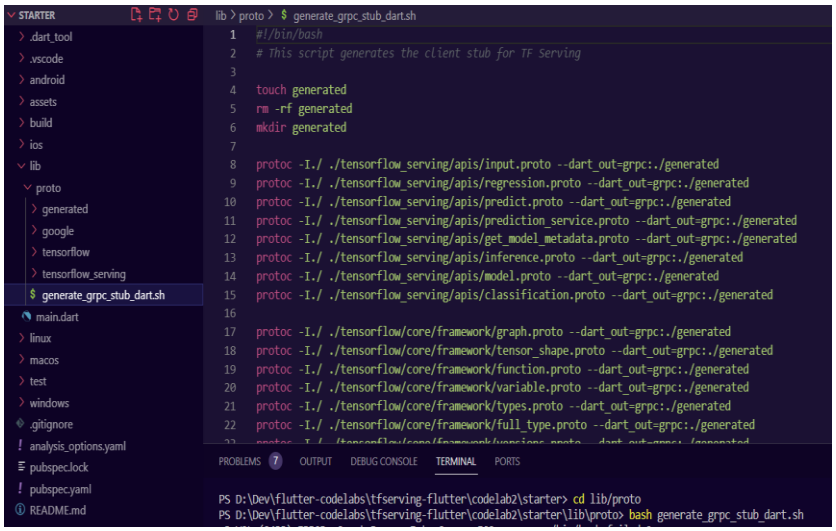
11. Connect the Flutter app with Tensorflow Serving through grPC

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.

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

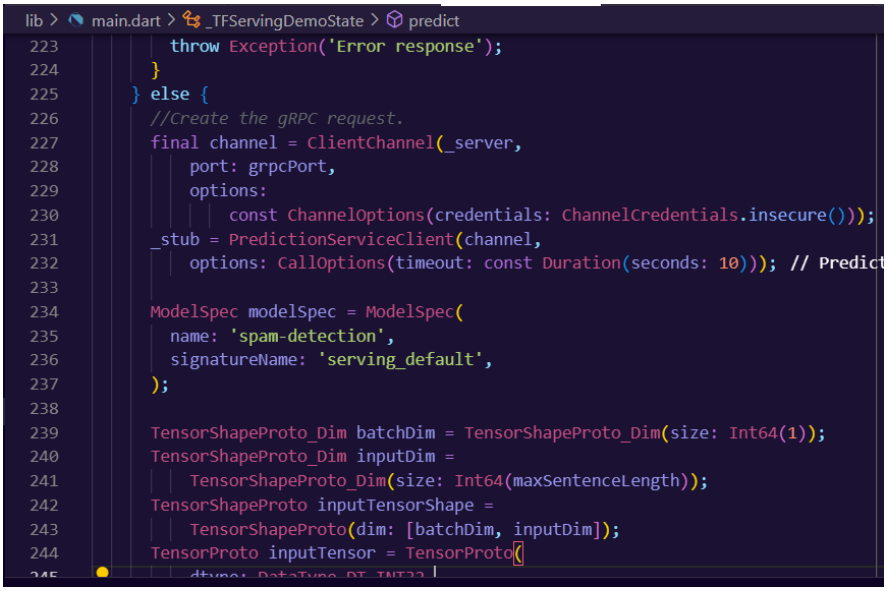
bash generate_grpc_stub_dart.sh



12. Here,I create the gRPC request

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

- Add this code to create the gRPC request:



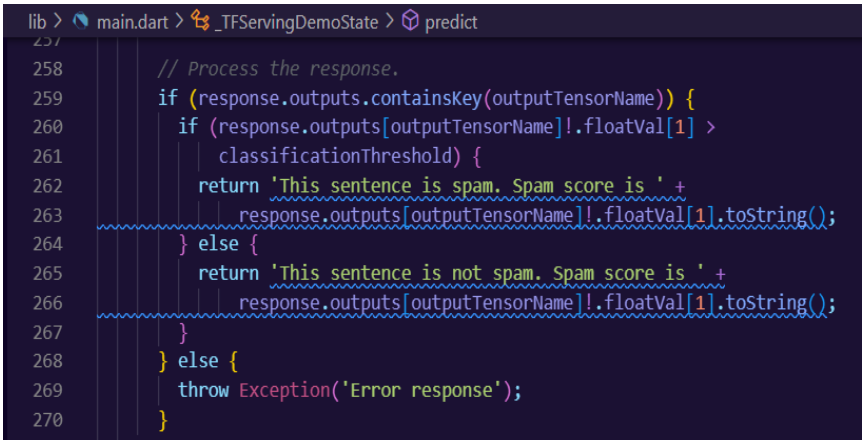
Note: The input and output tensor names could differ from model to model, even if the model architectures are the same. Make sure to update them if you train your own model.

Send the gRPC request to TensorFlow Serving

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

Process the gRPC response from TensorFlow Serving

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



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