https://towardsdatascience.com/spam-classification-in-android-with-tensorflow-lite-cde417e81260

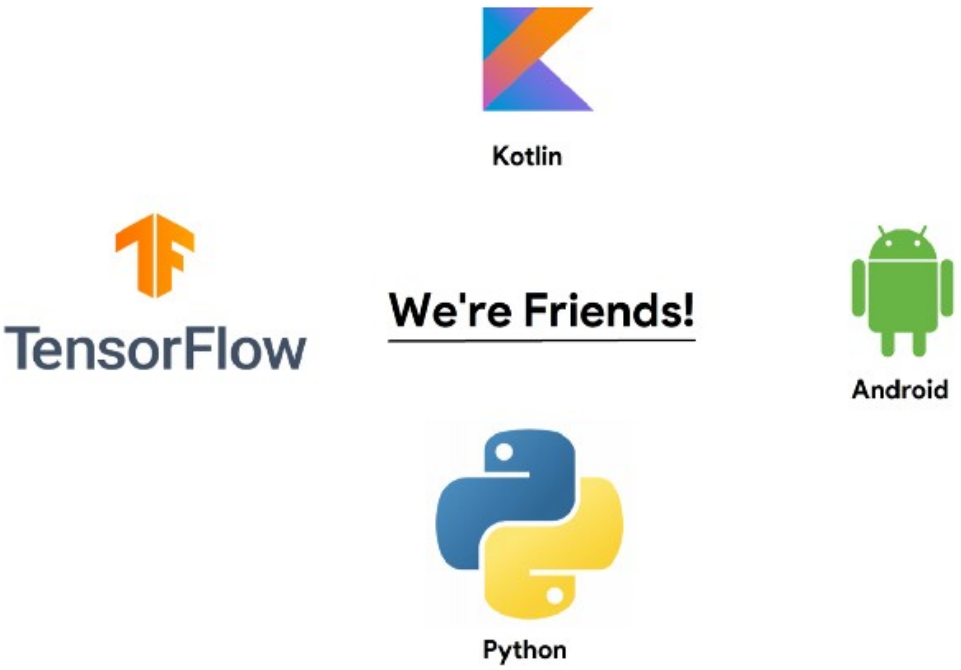
Shubham Panchal

May 1, 2019

MOBILE MACHINE LEARNING

Text Classification In Android With TensorFlow Lite

Classifying text with TF Lite models in Android



Machine learning has proved to be excellent in some of the use cases like spam classification which we'll perform in your Android application. We'll get started with it in Python, that's where we create our Classifier using Keras (TensorFlow).

> This article assumes that you have preliminary knowledge regarding TensorFlow, text classification and Android app development.

The RoadMap

1. ***We create a classifier in Python using TensorFlow and Keras.***
2. ***Convert the Keras model to a TensorFlow Lite model***
3. ***Convert the Keras' tokenizer vocabulary to a JSON file.***
4. ***Load the TF Lite model and JSON file in Android.***
5. ***Perform inference on the model in the Android app.***

I will be referring to various files of the GitHub repository of this project. You can open it alongside for a better learning experience.

Python Project Repo -> https://github.com/shubham0204/Spam_Classification_TF

Android Project Repo -> https://github.com/shubham0204/Spam_Classification_Android_Demo

Let's Start (With Python First!)

First, we need to parse the data from the CSV file using Pandas. Then we need to tokenize the messages, pad them and store them in NumPy arrays for training and testing. At last, we have the `DataProcessor.py` script generating 5 files. **(See DataProcessor.py file)**

```
android /
        - word_dict.json
processed_data /
        - x.npy
        - y.npy
        - test_x.npy
        - test_y.npy
```

Wondering from where the word_dict.json file came in? You know that we require a tf.keras.preprocessing.text.Tokenizer object to tokenize messages.

The Tokenizer maintains a Python dict object which has pairs for words and their indices. We convert this dict to a JSON file using :

> We can use the Tokenizer for converting texts to integer sequences. But, for inferencing in Android, we will need this vocabulary of words. Also, the code will output the max length of all the sequences. We will need this max length in Android.

We need to create a classifier in TensorFlow. If you are a seasoned ML developer, that's easy, right? **(See Model.py file)**

Note: ***activations.leaky_relu is a custom implementation and is not available in the official TensorFlow build. You can try using the LeakyReLU layer from tf.keras.layers.LeakyReLU.***

We can easily achieve 86% accuracy for the SMS Spam Collection Dataset by UCI Machine Learning on Kaggle.com. Next, we will save the trained Keras model to an h5py (.h5) file. We can do this by calling the method :

```
model.save( 'models/model.h5' )
```

Now, for this model to be used on Android, we need to convert this file (model.h5) to a TensorFlow Lite model. That's easy and also we can use post training quantization to reduce the model's size. **(See TFLiteBufferConverter.py file)**

The work in Python is complete. Now, we will only use the files listed under the android/ directory in Android. **(See the android/ directory)**

```
android/
    - word_dict.json
    - model.tflite
```

Creating the Android application (Kotlin Now!)

The Android application will classify the message as spam or not spam using the model we trained in Python. It will be a simple application with an EditText and a Button programmed in Kotlin .

Remember the GitHub Repo -

> https://github.com/shubham0204/Spam_Classification_Android_Demo

Starting with a Project

1. ***We create a simple Empty Project (most probably in Android Studio)***
2. ***Choose Kotlin/Java as the coding language.***
3. ***Copy all the files which were present under the `android/` directory in the Python project to your app's assets folder. (See `assets/` folder)***
4. ***Add the TensorFlow Lite (version 1.13.1) dependency in your `build.gradle` (app-level)***

```
dependencies {

    // Other app dependencies

    implementation 'org.tensorflow:tensorflow-lite:1.13.1'

}
```

In your `build.gradle` (app-level), add these lines which would disallow the compression of `.tflite` files.

```
Android {
      …
    }

    buildTypes {
        release {
            …
        }
    }
    aaptOptions {
        noCompress "tflite"
    }
}
```

Loading the vocabulary from the JSON file

We load the `word_dict.json` file from our app's assets folder. We will parse it into a `HashMap` which contains word-index pairs. **(See Classifer.kt file)**

Now, we can get the indices that we used for training in Python from the words using the `data` HashMap.

> Parsing such a huge JSON file on a Main (UI) thread isn't a good practice as it may disturb other processes on the thread. So, we use Kotlin Coroutines to perform this task efficiently.

Tokenizing and Padding the texts

This task was performed in Python. But, as we tend to feed the tokenized and padded integer sequences to our model, we will code two methods to do this in Kotlin. **(See Classifer.kt file)**

`this.maxlen` has a value of 171. Remember right? Our Python project told us that the max length was 171 and we padded all the sequences to a length of 171. `vocabData` is the HashMap, which we loaded in the previous method.

Note, if the word does not exist in the vocabulary (`data` HashMap) then we return an index of 0.

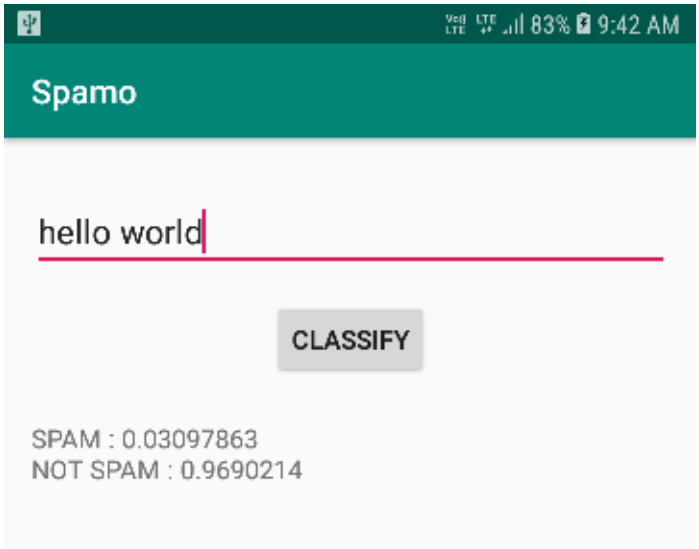
Loading the model from the assets folder

We load the model from the `assets` folder in the form of a MappedByteBuffer. **(See MainActivity.kt file)**

Classifying the message

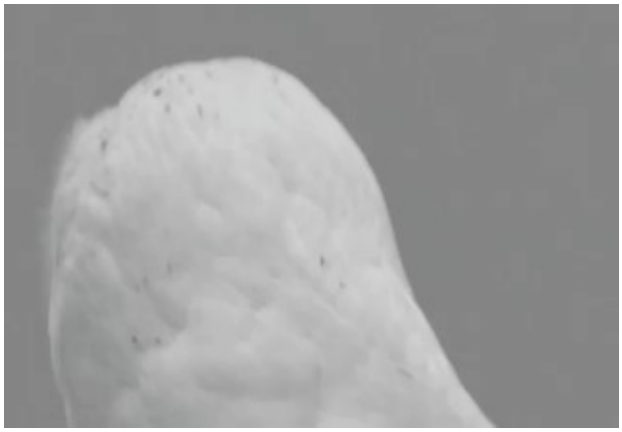
Now, we assemble all the methods together to classify the message and print the results. **(See MainActivity.kt file)**

Results



There’s more!

That’s All. Tired Right?



Hope you find the blog interesting. For any suggestions and queries, feel free to express them in the comments section.

TensorFlow and Android could just become

Thank You.