

Documento Tutorial

Herramientas de Aprendizaje Máquina para Sistemas Embebidos: Caso de TensorFlow Lite.

Brenda Valeria Lobo Mora

II Semestre 2022

Índice

1. Sobre este tutorial	3
2. Herramientas necesarias	3
3. Uso de TensorFlow	3
3.1. Entrenar un modelo de aprendizaje automático	3
4. Emulando la aplicación generada	7
4.1. Para Windows	7
4.2. Para Linux	9
5. Resultados de la simulación	10

1. Sobre este tutorial

En este tutorial se tiene como fin experimentar el uso de TensorFlow Lite como herramienta que permite generar un modelo de aprendizaje automático. La aplicación permite reconocer imágenes de dígitos escritos a mano con TensorFlow Lite para implementarlo en una aplicación de Android. Finalmente se utiliza un emulador de Android para ejecutar la aplicación creada.

2. Herramientas necesarias

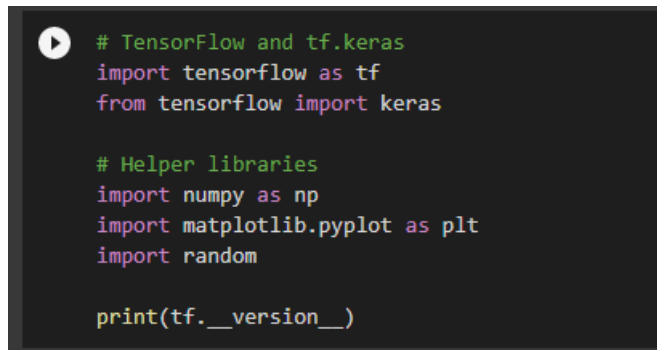
- Acceso a [Google Colab](#) o un entorno de Python con TensorFlow 2.0+
- Emulador de Android para Linux: [Genymotion](#), o bien, un Emulador de Android para Windows: [BlueStacks](#).
- Se utiliza Android Studio (v4.1+) para generar las apk, para el caso de este tutorial se brinda la aplicación ya lista para utilizar en el emulador encontrada en el Github [Minitaller](#)

3. Uso de TensorFlow

3.1. Entrenar un modelo de aprendizaje automático

Para entrenar el modelo se debe acceder al código disponible en Google Colab: [código](#)

1. Se deben importar los paquetes y las librerías necesarias para el procesamiento de datos y para la visualización de gráficos. (**Ejecutar código y mostrar la versión de TensorFlow.**)



```
# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
import random

print(tf.__version__)
```

Figura 1: Dependencias necesarias.

2. Se utiliza la base de datos MNIST que contiene 60000 imágenes de entrenamiento y 10000 imágenes de prueba de dígitos escritos a mano. Estos son agrupados entre los datos y normaliza la imagen de entrada para que el valor del pixel esté entre 0 y 1.

```
# Keras provides a handy API to download the MNIST dataset, and split them into
# "train" dataset and "test" dataset.
mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Normalize the input image so that each pixel value is between 0 to 1.
train_images = train_images / 255.0
test_images = test_images / 255.0
print('Pixels are normalized')
```

Figura 2: Código de división de imágenes y la normalización de píxeles.

3. Se muestran las primeras 25 imágenes de la base de datos. Cambie el código y cambie el rango por 9 (**Ejecute el código y tome una captura de pantalla de las 9 imágenes.**)).

```
# Keras provides a handy API to download the MNIST dataset, and split them into
# "train" dataset and "test" dataset.
mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Normalize the input image so that each pixel value is between 0 to 1.
train_images = train_images / 255.0
test_images = test_images / 255.0
print('Pixels are normalized')
```

Figura 3: Código que permite mostrar las imágenes.

4. Se entrena el modelo basado en la base de datos, se utiliza una red neuronal convolucional simple. Aquí se define el modelo de la arquitectura, el modo de entrenamiento y se entrena el modelo de clasificación de dígitos (**ejecutar el código y tomar una captura de pantalla de las precisiones obtenidas**).

```
# Define the model architecture
model = keras.Sequential([
    keras.layers.InputLayer(input_shape=(28, 28)),
    keras.layers.Reshape(target_shape=(28, 28, 1)),
    keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation=tf.nn.relu),
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation=tf.nn.relu),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),
    keras.layers.Dropout(0.25),
    keras.layers.Flatten(),
    keras.layers.Dense(10)
])

# Define how to train the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Train the digit classification model
model.fit(train_images, train_labels, epochs=5)
```

Figura 4: Código de entrenamiento del modelo.

5. Se va a editar la instancia `.fit` para obtener mayor precisión en los resultados. Primero, se va a cambiar el valor de **epochs=10**, que es la cantidad de iteraciones sobre la imagen, entre mas iteraciones mayor precisión. Además en la instancia `.fit` se va a agregar el argumento **use_multiprocessing=True**, este usa los hilos base del procesador, lo que aumenta el rendimiento en el entrenamiento de la precisión. **Tome una captura de pantalla de los nuevos resultados de precisión obtenidos.**

```
model.fit(train_images, train_labels, epochs=10, use_multiprocessing=True)
```

6. En la instancia `.fit`, también se puede colocar el argumento **validation_data**, que genera un punto al cual se desea llegar, cambia la perspectiva a aprendizaje de refuerzo. **Tome una captura de pantalla de los nuevos resultados de precisión obtenidos.**

```
model.fit(train_images, train_labels,
          epochs=10, use_multiprocessing=True, validation_data=(test_images,
                                                                test_labels))
```

7. Se evalúa el modelo utilizando las imágenes del set de datos de prueba. **Tome una captura de pantalla del resultado de la precisión obtenida.**

```
# Evaluate the model using all images in the test dataset.
test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)
```

Figura 5: Código para observar la precisión del modelo.

8. Se muestra el resultado del modelo, cambie el rango de observación con 50 imágenes, si se muestra un dato en rojo es porque la predicción es diferente a la etiqueta de la imagen. **Tome una captura de pantalla a las imágenes que se muestren al ejecutar el código.**

```
# A helper function that returns 'red'/'black' depending on if its two input
# parameter matches or not.
def get_label_color(val1, val2):
    if val1 == val2:
        return 'black'
    else:
        return 'red'

# Predict the labels of digit images in our test dataset.
predictions = model.predict(test_images)

# As the model output 10 float representing the probability of the input image
# being a digit from 0 to 9, we need to find the largest probability value
# to find out which digit the model predicts to be most likely in the image.
prediction_digits = np.argmax(predictions, axis=1)

# Then plot 100 random test images and their predicted labels.
# If a prediction result is different from the label provided label in "test"
# dataset, we will highlight it in red color.
plt.figure(figsize=(18, 18))
for i in range(100):
    ax = plt.subplot(10, 10, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    image_index = random.randint(0, len(prediction_digits))
    plt.imshow(test_images[image_index], cmap=plt.cm.gray)
    ax.xaxis.label.set_color(get_label_color(prediction_digits[image_index],\
                                             test_labels[image_index]))
    plt.xlabel('Predicted: %d' % prediction_digits[image_index])
plt.show()
```

Figura 6: Código para observar si coinciden las etiquetas con la predicción.

9. Se convierte el modelo de Keras a formato de TensorFlow Lite. **Tome captura de pantalla del tamaño del modelo generado.**

```
# Convert Keras model to TF Lite format.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_float_model = converter.convert()

# Show model size in KBs.
float_model_size = len(tflite_float_model) / 1024
print('Float model size = %dKBs.' % float_model_size)
```

Figura 7: Convierte modelo de Keras a TensorFlow Lite.

10. Como se implementa el modelo en un dispositivo móvil, se desea que el modelo sea lo más pequeño y rápido posible. La cuantificación es una técnica común que se usa a menudo en el aprendizaje automático. Ejecute el código para un número de 8 bits para aproximar los pesos de 32 bits, lo que a su vez reduce el tamaño del modelo en un factor de 4 (**Tome una captura de pantalla del tamaño del modelo**).

```
# Re-convert the model to TF Lite using quantization.
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_quantized_model = converter.convert()

# Show model size in KBs.
quantized_model_size = len(tflite_quantized_model) / 1024
print('Quantized model size = %dKBs,' % quantized_model_size)
print('which is about %d%% of the float model size.' \
      % (quantized_model_size * 100 / float_model_size))
```

Figura 8: Cambia el tamaño del modelo.

Seguidamente agregue la siguiente línea, antes de `tflite_float_model`:

```
converter.target_spec.supported_types = [tf.float16]
```

Observe el cambio, y tome una captura de pantalla del tamaño del modelo.

11. Se compara el modelo Keras con el modelo generado de TensorFlow Lite y se observa la precisión. Normalmente dado que se quiere hacer el modelo más pequeño se disminuye la precisión.

```
Evaluate the TensorFlow Lite model

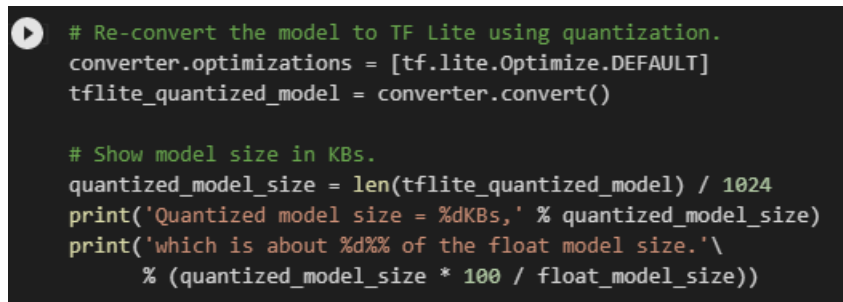
By using quantization, we often traded off a bit of accuracy for the benefit of having a significant drop of our quantized model.

# A helper function to evaluate the TF Lite model using "test" dataset.
def evaluate_tflite_model(tflite_model):
    # Initialize TFLite interpreter using the model.
    interpreter = tf.lite.Interpreter(model_content=tflite_model)
    interpreter.allocate_tensors()
    input_tensor_index = interpreter.get_input_details()[0]["index"]
    output = interpreter.tensor(interpreter.get_output_details()[0]["index"])

    # Run predictions on every image in the "test" dataset.
    prediction_digits = []
```

Figura 9: Código de comparación entre modelos (ejecute el código).

12. Finalmente se descarga el modelo con la extensión .tflite.



```
# Re-convert the model to TF Lite using quantization.
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_quantized_model = converter.convert()

# Show model size in KBs.
quantized_model_size = len(tflite_quantized_model) / 1024
print('Quantized model size = %dKBs,' % quantized_model_size)
print('which is about %d%% of the float model size.'\
      % (quantized_model_size * 100 / float_model_size))
```

Figura 10: Descarga del modelo generado.

Para efectos de este tutorial, se genera una aplicación, llamada **app_debug.apk** tiene solo tiene 5 iteraciones de entrenamiento y no se agregan las características de mejora descritas anteriormente. Esta aplicación se encuentra en el Github [Minitaller](#). Se puede hacer su propia aplicación siguiendo el tutorial de [digit_classifier](#), sin embargo, no es tomado como parte de este tutorial.

4. Emulando la aplicación generada

4.1. Para Windows

Puede utilizar el emulador de Android para Windows: [BlueStacks](#). Son necesarios 5GB de espacio libre en disco y al menos tener 4 GB de RAM.

Al ejecutar el archivo .exe descargado, se procede a hacer la instalación. Una vez instalado, se tiene que desplegar una pantalla como la Fig. 11, y se debe ubicar abrir el archivo .apk (flecha roja).

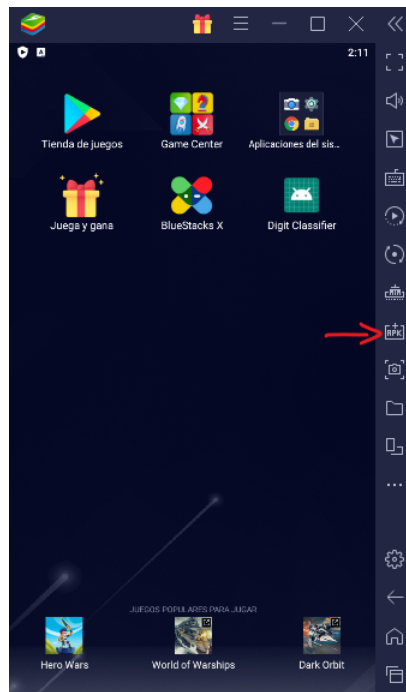


Figura 11: Emulador de BlueStacks (flecha roja indica de dónde abrir el archivo .apk).

Seguidamente, aparece la aplicación Digit Classifier, y ahí se debe escribir un número. En la parte inferior se observa la precisión.

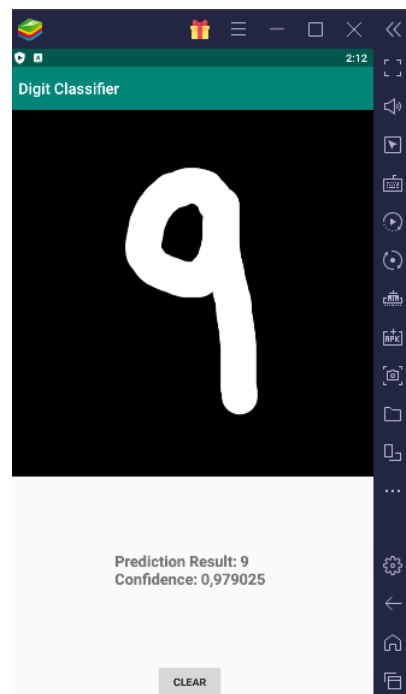


Figura 12: Ejecutando la aplicación.

4.2. Para Linux

Puede utilizar el emulador de Android para Linux: [Genymotion](#)

Para utilizar Genymotion Desktop, debe crear y activar una cuenta de Genymotion en [cuenta](#). Debe tener mínimo 120 MB para Genymotion + 1 GB por dispositivo virtual. La duración de descarga es de aproximadamente 5 min.

Se debe instalar un dispositivo virtual, se puede escoger como se muestra en la siguiente imagen:

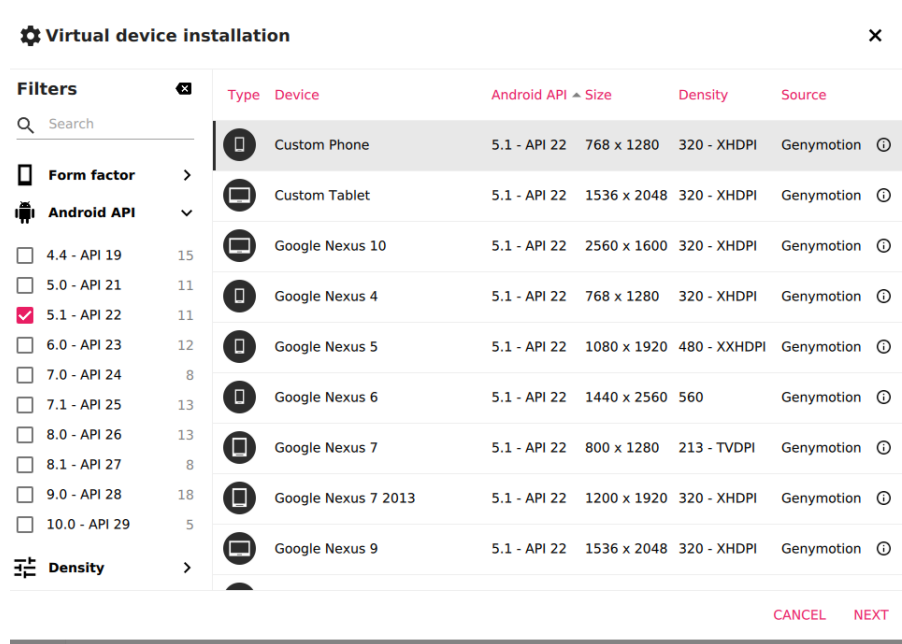


Figura 13: Configuración de la aplicación Genymotion.

The image shows the Genymotion configuration interface. It has a light gray background with sections separated by horizontal lines. The sections are:
1. **Name**: A text input field containing 'Telefono'.
2. **Display**: Contains three options: 'Predefined' (selected with a red radio button), 'Custom' (unselected), and 'Start in full-screen mode' (checkbox). To the right of 'Predefined' are two dropdown menus showing '768 x 1280' and '320 - XHDPI'.
3. **System**: Contains three settings: 'Android version' set to '5.1', 'Processor(s)' set to '4', and 'Memory size' set to '1024'.
4. **Android system options**: Contains 'Show Android navigation bar' which is checked with a red checkbox.
At the bottom right, there are two red buttons: 'BACK' and 'INSTALL'.

Name	
Telefono	

Display	
<input checked="" type="radio"/> Predefined	768 x 1280 320 - XHDPI
<input type="radio"/> Custom	
<input type="checkbox"/> Start in full-screen mode	

System	
Android version	5.1
Processor(s)	4
Memory size	1024

Android system options	
Show Android navigation bar	<input checked="" type="checkbox"/>

BACK INSTALL

Figura 14: Configuraciones de Genymotion.

Se debe arrastrar la apk a la interfaz de dispositivo móvil.

5. Resultados de la simulación

Una vez que se tenga el emulador ejecutando la aplicación, se debe **tomar una captura de pantalla de la aplicación escribiendo tres veces el número 9**. Observar la precisión de la aplicación. **Realizar lo anterior con dos números diferentes de su preferencia.**

Referencias

- [1] <https://colab.research.google.com/>
- [2] <https://www.genymotion.com/download/>
- [3] <https://www.bluestacks.com/es/bluestacks-5.html>
- [4] https://colab.research.google.com/github/tensorflow/examples/blob/master/lite/codelabs/digit_classifier/ml/step2_train_ml_model.ipynb
- [5] <https://developer.android.com/codelabs/digit-classifier-tflite#2>