MNIST for STM32

Marc Vernet

December 6, 2021

1 Introduction

The goal of this assignment was to develop the code required to be able to use an STM32 board as an MNIST predictior.

The STM32 board contains a touchscreen trough which the user is able to write a digit. The board also contains two buttons, and when the blue push-button is pressed, the digit written is analysed by a neural network and the predicted output is showed on the screen.

The neural networks used for the MNIST prediction are tensorflow models compressed with tensorflow lite in order to be used on the embedded device. Three different models are used, and the code can be seen in the Models section. The three models are very simple, consisting of different configurations of dense layers and dropouts.

2 Results

Considering that the models used are very simple, the results are quite good. There's very little difference or improving between the models.

Among all the digits seem that 1 and 6 are the most difficult to predict and they are often confused with other digits. The third model seems to solve the confusion with 6, but makes worse predictions with 1, so it's not clear if the added complexity is helping with accuracy.

Possibly a convolutional neural network could bring better results, but it seems that there is little space for improvements.

digit	model1	model2	model3
0	always correct	always correct	always correct
1	most of the time correct, sometimes confuses with 9	most of the time correct, sometimes confuses with 4	confuses with 9 very often
2	always correct	always correct	always correct
3	always correct	always correct	always correct
4	always correct	always correct	always correct
5	always correct	always correct	always correct
6	most of the time correct, sometimes confuses with 6	confuses often with 5 and 2	always correct
7	always correct	always correct	always correct
8	always correct	always correct	always correct
9	always correct	always correct	always correct

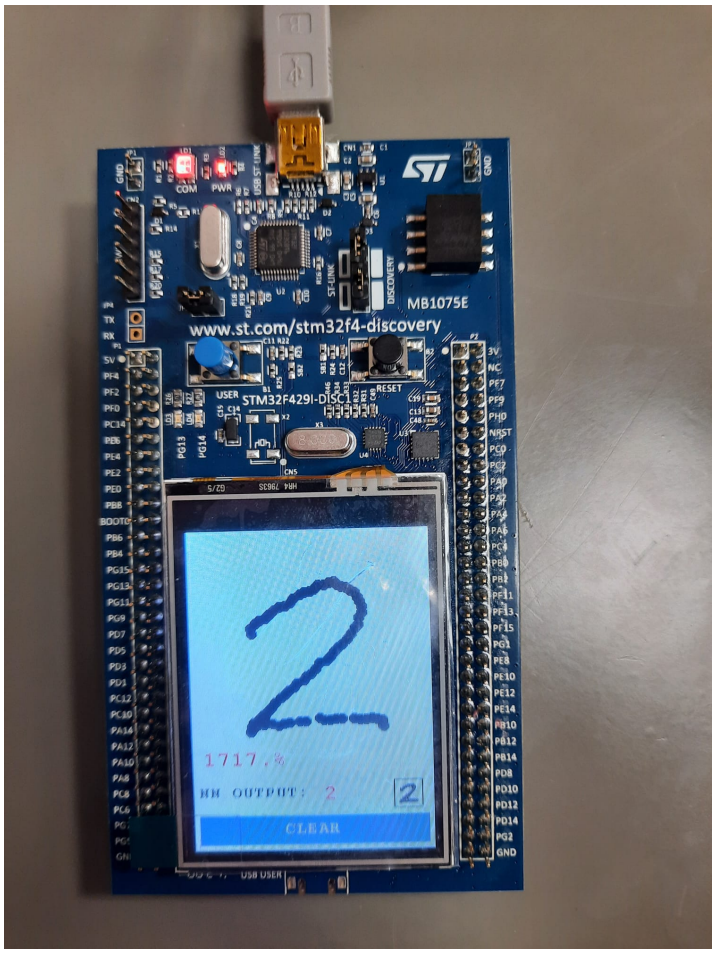

Figure 1: stm32 board with a digit written

3 Models

Listing 1: Model 1

```
def model1():
  return tf.keras.models.Sequential([
      tf.keras.layers.Flatten(input_shape=(28, 28)),
      tf.keras.layers.Dense(80, activation='elu'),
      tf.keras.layers.Dense(60, activation='elu'),
      tf.keras.layers.Dropout(0.2),
      tf.keras.layers.Dense(10)
  ])
```

Listing 2: Model 2

```
def model2():
  return tf.keras.models.Sequential([
      tf.keras.layers.Flatten(input_shape=(28, 28)),
      tf.keras.layers.Dense(80, activation='elu'),
      tf.keras.layers.Dense(60, activation='elu'),
      tf.keras.layers.Dense(40, activation='elu'),
      tf.keras.layers.Dense(20, activation='elu'),
      tf.keras.layers.Dropout(0.2),
      tf.keras.layers.Dense(10)
  ])
```

Listing 3: Model 3

```
def model3():
  return tf.keras.models.Sequential([
      tf.keras.layers.Flatten(input_shape=(28, 28)),
      tf.keras.layers.Dense(80, activation='elu'),
      tf.keras.layers.Dropout(0.2),
      tf.keras.layers.Dense(60, activation='elu'),
      tf.keras.layers.Dropout(0.2),
      tf.keras.layers.Dense(40, activation='elu'),
      tf.keras.layers.Dropout(0.2),
      tf.keras.layers.Dense(20, activation='elu'),
      tf.keras.layers.Dropout(0.2),
      tf.keras.layers.Dense(10)
  ])
```