AN12603

Handwritten Digit Recognition Using TensorFlow Lite Micro on i.MX RT devices

Rev. 1 — 19 October 2021 Application Note

Contents

1 Introduction

This application note focuses on handwritten digit recognition on embedded systems through deep learning. It explains the process of creating an embedded machine learning application that can classify handwritten digits and presents an example solution based on NXP's SDK and the eIQ™ technology.

Handwritten digit recognition with models trained on the MNIST data set is a popular "Hello World" project for deep learning as it is simple to build a network that achieves over 90% accuracy for it. There are also many existing open source implementations of MNIST models on the Internet, making it a well-documented starting point for machine learning beginners.

The MNIST eIQ example consists of several parts. The digit recognition is performed by a TensorFlow Lite model, with an architecture similar to LeNet-5 (LeCun, LeNet-5, convolutional neural networks, 2019), which was converted from the TensorFlow implementation released by Google. The GUI was created in Embedded Wizard Studio and uses the Embedded Wizard library. The model allocation, input, and output processing and inference are handled by the SDK and custom code written specifically for the example.

2 MNIST data set

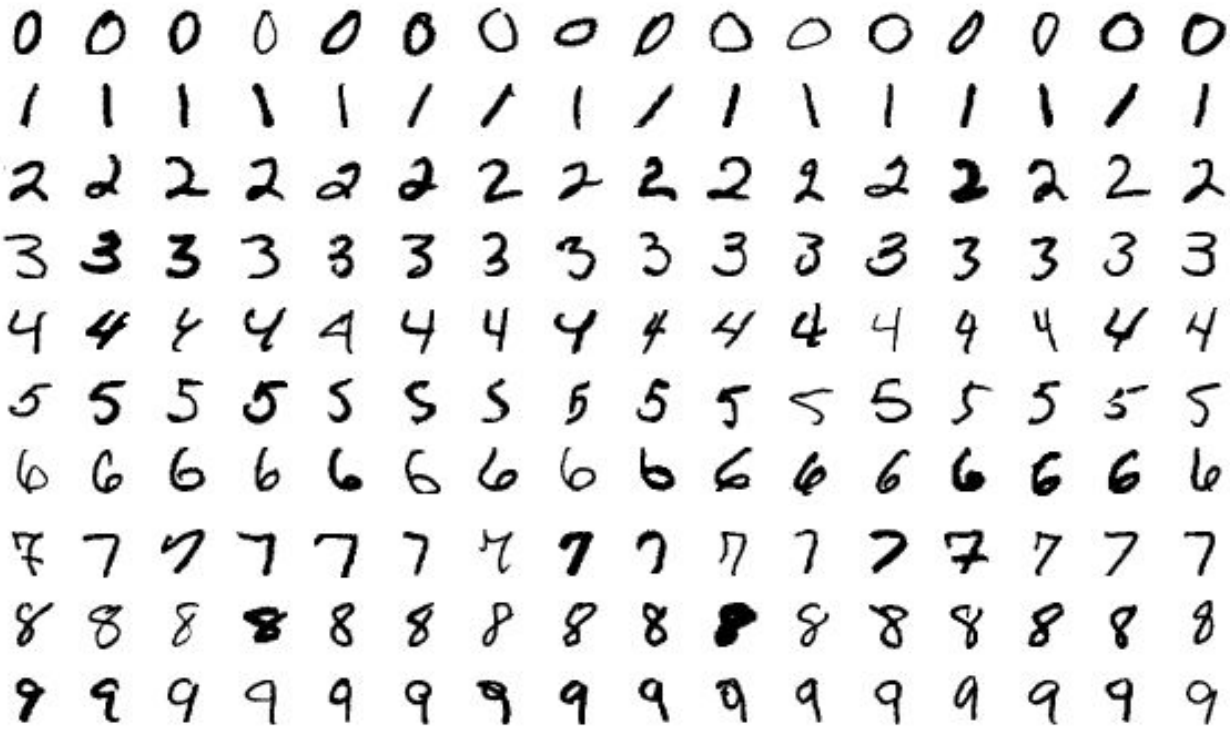

Figure 1. MNIST dataset example (Steppan, 2017)

The dataset contains centered grayscale 28x28 images of handwritten digits like in Figure 1. It consists of 60000 training examples and 10000 testing examples. It was collected from high school students and Census Bureau employees and is a subset of a larger set available from NIST. The dataset was selected and published by Yann LeCun, Corinna Cortes, and Christopher J.C. Burges and is open source (LeCun, The Mnist Database, 2019) . The dataset has been used to benchmark different machine learning algorithms and while convolutional neural networks typically give the best results, there are other viable approaches as well. Among them are support vector machines (SVM), k-nearest neighbors algorithms (K-NN) and various types of neural networks. A survey of the different results was published in the Applied Sciences journal by MDPI in August 2019 (Baldominos, Saez, & Isasi, 2019). Even simple convolutional neural networks can achieve an accuracy of around 99%. Therefore, TensorFlow Lite was a suitable option for this task.

3 TensorFlow

TensorFlow is an open source cross-platform deep learning library developed at Google Brain. It is the most popular deep learning framework and is widely used in production both at Google and other large organizations. It is available through a low-level python API, which is useful for skilled and experienced developers or through other, higher-level libraries, like Keras. Keras is simpler, beginner friendly, and enables anyone to try and learn about machine learning. TensorFlow is supported by a very large user community and by official documentation, guides, and examples from Google.

To enable TensorFlow on mobile and embedded devices, Google developed the TensorFlow Lite framework. It gives these computationally restricted devices the ability to run inference on pre-trained TensorFlow models that were converted to TensorFlow Lite. These converted models cannot be trained any further but can be optimized through techniques like quantization and pruning. However, TensorFlow Lite does not support all the original TensorFlow's operations and developers must keep that in mind when creating models.

The next evolution of TensorFlow is the TensorFlow Lite Micro, which is focused on microcontrollers. TensorFlow Lite Micro is a subset of TensorFlow Lite and is being developed by Google in tight collaboration with Arm. The library is further optimized by NXP with device specific optimizations to achieve even better performance. Since TensorFlow Lite Micro supports only TensorFlow Lite models, the process of training a model and then converting it remains the same. The differences are in the more optimized library and a slightly more limited list of supported operations. However, this list is growing over time and there should be no issues with running any models useful for embedded applications.

4 MNIST model

The model implementation chosen for this example is available on GitHub as one of the official TensorFlow models under the Apache 2.0 license. It is written in python and uses the built-in Keras library. The script builds a convolutional neural network that can achieve over 99% accuracy on the test set examples from the MNIST dataset. The TensorFlow Lite graph can be seen in Figure 2. This graph was generated with Netron (Roeder, 2021), which is a visualizer for neural networks, deep learning and machine learning models. It supports many formats from different frameworks, including TensorFlow Lite, Caffe, Keras, and ONNX. For example, it can be used to display a neural network topology in a web browser and inspect the individual layers, operations and connections used in the model.

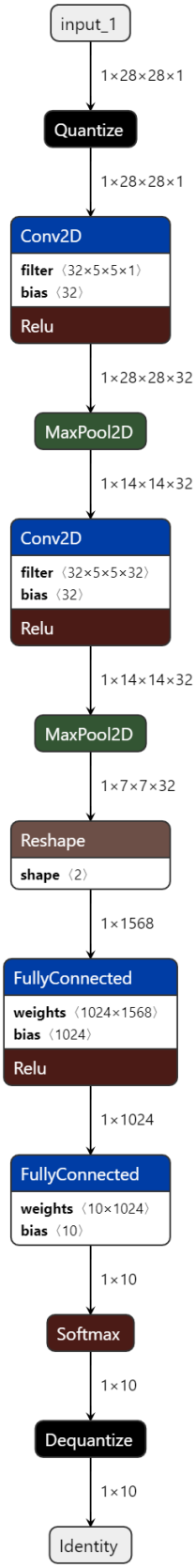

Figure 2. Model visualization in netron (TensorFlow Lite)

See the below code for the Keras model definition:

```
def create_model():
image = tf.keras.layers.Input(shape=(28, 28, 1))

y = tf.keras.layers.Conv2D(filters=32,
kernel_size=5,
padding='same',
activation='relu')(image)
y = tf.keras.layers.MaxPooling2D(pool_size=(2, 2),
strides=(2, 2),
padding='same')(y)
y = tf.keras.layers.Conv2D(filters=32,
kernel_size=5,
padding='same',
activation='relu')(y)
y = tf.keras.layers.MaxPooling2D(pool_size=(2, 2),
strides=(2, 2),
padding='same')(y)
y = tf.keras.layers.Flatten()(y)
y = tf.keras.layers.Dense(1024, activation='relu')(y)
y = tf.keras.layers.Dropout(0.4)(y)

probs = tf.keras.layers.Dense(10, activation='softmax')(y)

model = tf.keras.models.Model(image, probs, name='mnist')
return model
```

The trained model was converted to TensorFlow Lite using the TensorFlow converter API. For details, see https://www.tensorflow.org/api_docs/python/tf/lite/TFLiteConverter. An example script for this purpose can be found in the AN12603 software package available in the documentation tab for eIQ and the relevant RT devices at NXP website. The script is called model_converter.py. For compatibility purposes with the current (August 2021) version (2.4.1) of the TensorFlow Lite Micro library used in NXP's SDK, the 2.4.1 version of TensorFlow was used for training and converting the model.

Lastly, the xxd utility was used to convert the TensorFlow Lite model into a binary array that could be loaded by the SDK application. The conversion process is described in more detail in the eIQ User Guides.

xxd -i converted_model.tflite converted_model.h

After converting the model, the output header file needs a few changes before it is ready for use.

```
#ifndef __XCC__
#include <cmsis_compiler.h>
#else
#define __ALIGNED(x) __attribute__((aligned(x)))
#endif

#define MODEL_NAME "lenet_mnist"
#define MODEL_INPUT_MEAN 0.0f
#define MODEL_INPUT_STD 255.0f

static const uint8_t model_data[] __ALIGNED(16) = {
  0x20, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x12, 0x00, 0x1c, 0x00, 0x04, 0x00, 0x08, 0x00, 0x0c, 0x00,
  0x10, 0x00, 0x14, 0x00, 0x00, 0x00, 0x18, 0x00, 0x12, 0x00, 0x00, 0x00,
  0x03, 0x00, 0x00, 0x00, 0x88, 0x38, 0x19, 0x00, 0x58, 0x22, 0x19, 0x00,
  0x40, 0x22, 0x19, 0x00, 0x3c, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
```

Figure 3. Model header (beginning)

```
  0x09, 0x00, 0x00, 0x00, 0x0c, 0x00, 0x0c, 0x00, 0x07, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x08, 0x00, 0x0c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x16,
  0x16, 0x00, 0x00, 0x00, 0xf0, 0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x11,
  0x02, 0x00, 0x00, 0x00, 0x11, 0x00, 0x00, 0x00, 0x0c, 0x00, 0x10, 0x00,
  0x07, 0x00, 0x00, 0x00, 0x08, 0x00, 0x0c, 0x00, 0x0c, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x03, 0x03, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00
};
unsigned int model_data_len = 1653072;
```

Figure 4. Model header (end)

xxd is a hexdump utility (Weigert, Nugent, & Moolenaar, 2019) that can be used to convert back and forth between the hex dump and binary form of a file. In this case, the utility is used to convert the tflite binary into a C/C++ header file that can be added to an eIQ project.

5 Embedded wizard studio

Embedded Wizard Studio (TARA Systems GmbH, 2021) is an IDE for developing graphical user interfaces for embedded devices. It is offered in three tiers with different levels of support and pricing. One of them is the free tier, which can be used for evaluation and prototyping purposes, limits the project complexity and adds a watermark over the GUI. The free tier was more than enough for the MNIST demo, as the created graphics reached only 10% of the maximum complexity allowed. One of the advantages of the IDE is its ability to generate MCUXpresso projects based on NXP's SDK. It means that after creating the GUI in the IDE, the developer can immediately test it on their device. All of the projects created for this application note are part of the AN12603 software package.

The IDE offers a wide variety of GUI objects and tools, including buttons, touch input areas, shapes, graphics, triggers that can react to button presses or screen touches and many more. Placing them on a canvas and setting their properties to fit the developers needs is intuitive and user-friendly and largely speeds up the GUI development process.

Several steps had to be performed to merge the GUI project with the eIQ application project. Since the generated project is written in C and the eIQ examples are in C/C++, the ewmain.h header file needs to have its contents surrounded by:

```
#ifdef __cplusplus
extern "C" {
#endif
/* C code */
#ifdef __cplusplus
}
#endif
```

A new embedded wizard folder had to be created in the base eIQ project for the generated source files. Several additional drivers had to be added to the base project, the pin_mux files had to be replaced with the Embedded Wizard generated ones and some additional differences had to be merged in the board files and timer files. Lastly, the references, include paths, symbol definitions, and memory configurations had to be adjusted, so that everything could be compiled together.

6 Application functionality

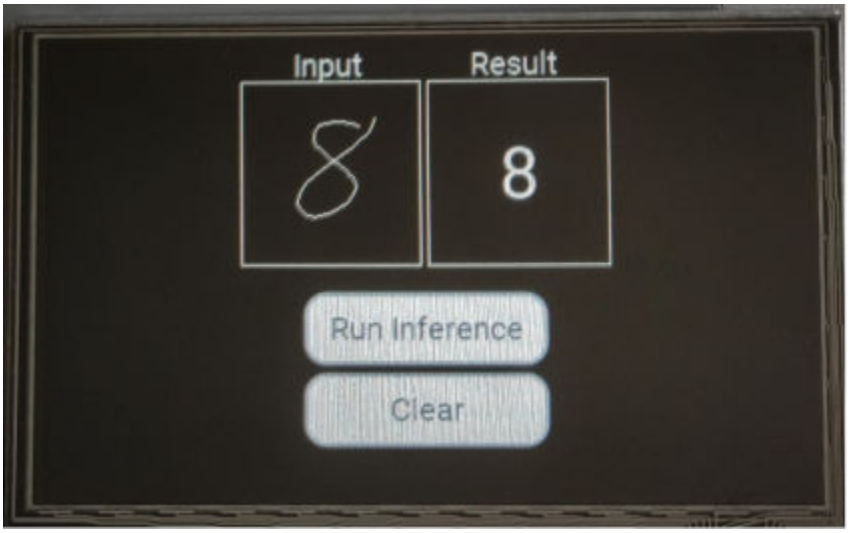

Figure 5. Example inference test

The application is controlled through a GUI displayed on a touch sensitive LCD. The GUI, as shown in Figure 5, includes a touch-based input area for writing digits, an output area for displaying the results of inference and two buttons, one for running the inference and the other for clearing the input and output areas. It also outputs the result and the confidence of the prediction to standard output, which can be read by using programs like PuTTY and listening on the associated COM port while the board is connected to the PC.

7 Accuracy

Figure 6. USA style numbers (Wagner, 2011)

As the MNIST dataset is written by people from the USA, the application correctly recognizes single digits written in the USA style of handwritten numbers shown in Figure 6. However, mainland European countries, for example, tend to write several of the numbers differently, as apparent in Figure 7, and these styles can lead to wrong predictions.

Figure 7. Mainland Europe style numbers (Wagner, 2011)

Some countries write the 1 with the left line shorter, about half or third the length of the straight line. These differences can confuse the machine leaning model and make it classify a European 1 as a USA 7, since they are so similar in shape. Another important aspect influencing accuracy is the difference between how the application gets its input and how the images in data sets are taken. Even though the model can achieve over 99% accuracy on the training and testing data sets, it is not as accurate when used in the application. This is because digits written on an LCD with a finger are never same as digits written on a paper with a pen. Finally, the application input is read as white and black pixels without any distortions caused by compression. On the other

hand, the MNIST data set contains images that are gray scale and do suffer from compression. It highlights the importance of training production models on real production data. In order to achieve better results, a new data set composed of digits written by people from all over the world would have to be collected. Additionally, the means of input would have to be the same as in the digit recognition application. The current application could be adjusted to save the input numbers instead of sending them to the model for recognition. To retrain the model afterward, transfer learning could be applied. The goal of this technique is to take a pretrained model, disable changes in some or even all the layers except the final ones and train it on a similar but different data set. Transfer learning produces very accurate models that are trained faster and require smaller data sets than regular training would need. The NXP Community website contains a walk-through of using TensorFlow Lite for Microcontrollers including the transfer learning technique (https://community.nxp.com/t5/eIQ-Machine-Learning-Software/Getting-Started-with-TensorFlow-Lite-for-Microcontrollers-on-i/ta-p/1124103) to retrain a model from classifying general categories of images to recognizing a small set of flowers.

It is recommended to also go through a follow-up the application note AN12892 (https://community.nxp.com/t5/eIQ-Machine-Learning-Software/Transfer-Learning-and-the-Importance-of-Datasets/ta-p/1109248), which focuses on transfer learning and data sets. This application note describes the process of collecting a new data set at one of the NXP sites and using this data set to retrain the model used here, while the original model achieves about 67% on the custom NXP validation data set, the retrained model achieves over 96% accuracy instead. Both the original and the retrained models are also included in the software package for AN12603 for readers to try and compare.

8 Implementation details

Embedded Wizard uses the so-called slots as triggers that react to GUI interactions. In the example, one of these slots is connected to the touch sensitive input area as an “on drag” trigger. When a user drags their finger over the area, the slot continually draws a single-pixel wide line under the finger. The drawing uses the color defined by the main color constant and is constrained to the input area.

The buttons also have slots assigned to them. The Clear button’s slot simply sets the color of pixels inside both the input and result areas to the background color. The Run Inference button’s slot saves references to the input area, the underlying bitmap, and the width and height of the area, and then passes them to a native C code, which processes the input image.

To make using the application more comfortable, the input area was created as a 112x112 square for RT1060 and 224x224 for RT1170. However, the actual input image for the machine learning model must be 28x28 pixels large. Since the line used for drawing is only one pixel wide and cannot be made any wider due to the technique used to draw it, additional preprocessing is necessary, otherwise scaling the image down would distort the input too much.

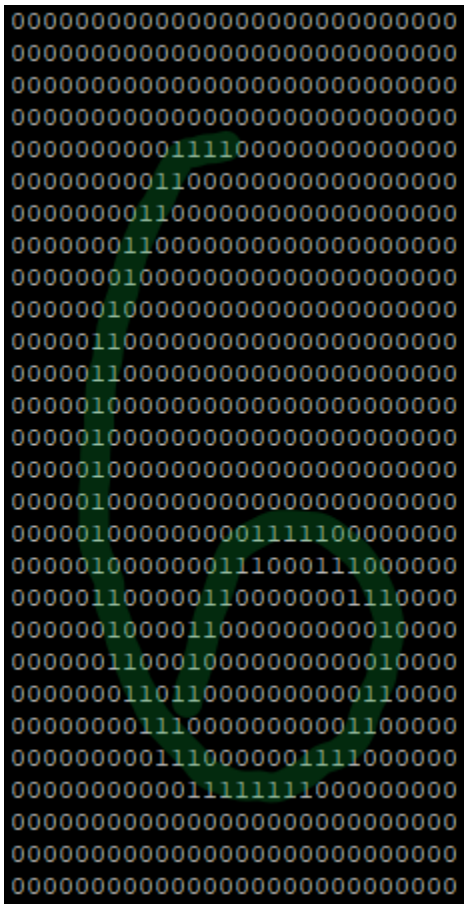

Figure 8. Preprocessed input logging representation (white pixels printed as “1”, black pixels as “0”)

To skip grayscale conversions, pixels of the main color, regardless of what it is, are considered white and everything else black. First, an array of 8-bit integers with the width and height of the input area is created and filled with zeroes. Afterwards, the image and array are iterated over, and every white pixel in the image is stored as 0xFF in the array. Additionally, each pixel is expanded into a "plus sign" with several pixel long lines, based on configuration, thickening the line in the process. Doing so makes downscaling the image much safer. When iterating over the input image, the loops must skip the irrelevant pixels outside the input area. The amount of this is always the same, since the pixels are stored continuously by row from left to right. Once the input is extracted, the drawing is cropped and centered to resemble the format of the MNIST images a bit more and then scaled to the 28x28 resolution. Additionally, the input needs to be rotated on RT1170 due to the whole GUI being rotated in the vertical display. Figure 8 shows an interpretation of the result, where the white pixels are represented by “1”s and black pixels by “0”s.

When the application starts, the machine learning model is allocated, loaded, and prepared for inference. Every time inference is requested, the model’s input tensor is loaded with the preprocessed input and passed to the model. Since the model expects float values and the functions used for image-processing work with 8-bit integers, the input must be copied into the tensor pixel by pixel and converted in the process. The inference result is written out both to standard output and the output area in the GUI.

The preprocessing is performed partially in “embeddedwizard/Application.c” and then finished in “source/model/input_proc.cpp”. The inference and results are handled in “source/mnist.cpp” as well. In “embeddedwizard/CustomConstant.c”, the main and background color constants are defined.

9 Extending the application example

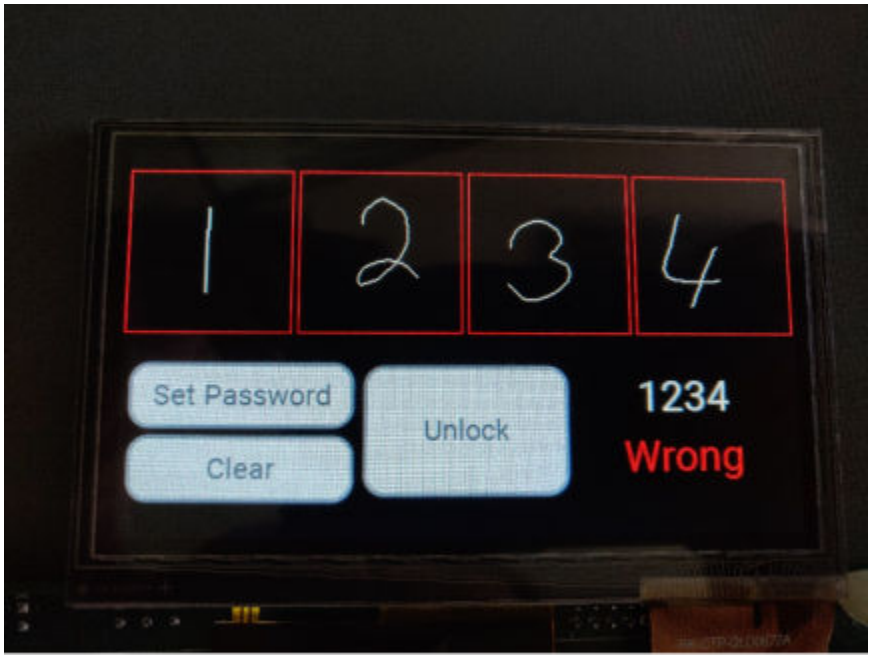

Figure 9. Wrong pin

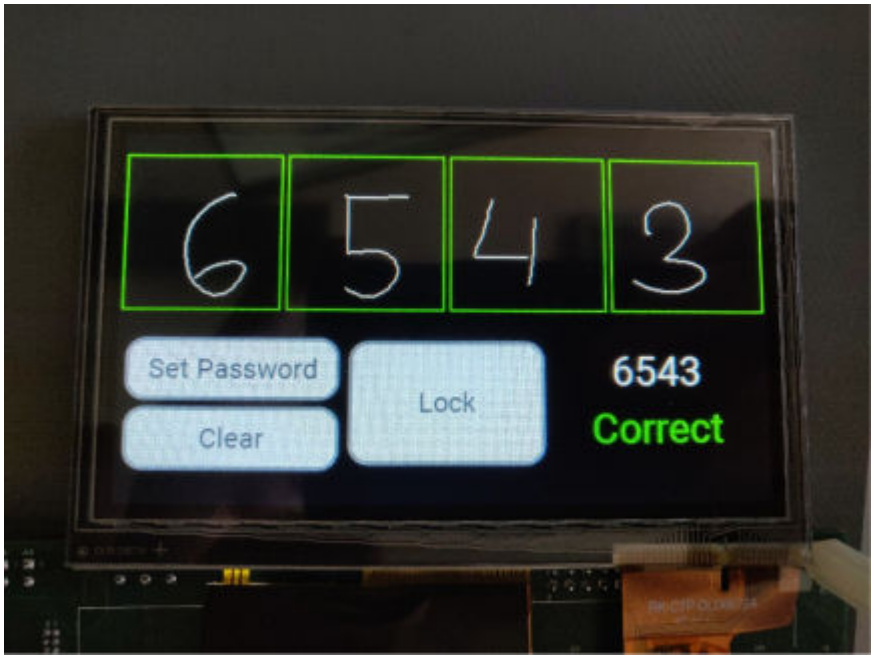

Figure 10. Correct pin

The simple example can be extended in several ways. For this application note, a simple digital lock with a 4-digit pin code as shown in Figure 9 and Figure 10 was implemented.

To achieve this, the whole input area is cloned three times and the processing was adjusted to include all four windows and run inference on each of the inputs one by one. In this extended version, inference gets triggered every time the lock is locked and the Unlock button is pressed or whenever the Set Password button is pressed. Functionality of the Clear button is similarly extended

to clean all four inputs and the text output in the bottom right corner. The Set Password button simply stores the current inputs as the pin for the lock in an integer array.

10 Conclusion

This application note introduced the problem of recognizing handwritten digits using machine learning algorithms and presented a viable solution on embedded platforms using TensorFlow Lite. The solution is developed as an application for the RT1060-EVK and RT1170-EVK. The document also shows how the achieved machine learning capabilities can be used for more complex scenarios, like a digital lock.

While the showcased application is simple in nature, it can serve as an introduction into machine learning on embedded devices. The potential of machine learning and AI on NXP embedded platforms is continuously improving as eIQ keeps getting even more advanced and optimized.

In the future, the digit recognition application and this application note will be extended to new releases in the i.MX RT Series.

11 References

- LeCun, Y. (2019). *LeNet-5, convolutional neural networks*: http://yann.lecun.com/exdb/lenet/
- LeCun, Y. (2019). *The Mnist Database*: http://yann.lecun.com/exdb/mnist/
- Steppan, J. (2017, December 14). *Sample images from MNIST test dataset*: https://upload.wikimedia.org/wikipedia/commons/2/27/MnistExamples.png
- TARA Systems GmbH. (2021): https://www.embedded-wizard.de/
- TensorFlow. (2021). TensorFlow Image Classification Repository: https://github.com/tensorflow/models/tree/master/official/vision/image_classification/
- Wagner, D. J. (2011, April 27). *Numbers and Counting: American vs. French*: https://ielanguages.com/blog/numbers-and-counting-american-vs-french/
- Baldominos, A., Saez, Y., & Isasi, P. (2019). A Survey of Handwritten Character Recognition with MNIST and EMNIST. Applied Sciences. 9, 3169. doi:10.3390/app9153169
- Roeder, L. (2021). Netron: https://github.com/lutzroeder/netron
- NXP. (2021). MCUXpresso Software Development Kit (SDK): https://mcuxpresso.nxp.com/
- TensorFlow. (2021). TensorFlow Lite converter: https://www.tensorflow.org/lite/convert
- Weigert, J., Nugent, T., & Moolenaar, B. (2019). xxd(1) - Linux man page: https://linux.die.net/man/1/xxd
- Huereca, A. (2021). eIQ Transfer Learning Lab with i.MX RT: https://community.nxp.com/docs/DOC-343827

12 Revision history

Table 1. Revision history

Revision number	Date	Substantive changes
1	19 October 2021	• Updated the associated software and added support for RT1170-EVK in sections; TensorFlow, MNIST model, Embedded wizard studio, Accuracy, Implementation details, Conclusion, and References

Table continues on the next page...

Table 1. Revision history (continued)

Revision number	Date	Substantive changes
		• Removed the "Memory Footprint" section
0	20 April 2020	Initial release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Limited warranty and liability — Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX,EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE, VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 19 October 2021
Document identifier: AN12603