AN12603

Handwritten Digit Recognition Using TensorFlow Lite on RT1060

Rev. 0 — 20 April 2020

Application Note

Contents

1 Introduction..... 1
2 MNIST Dataset..... 2
3 TensorFlow..... 2
4 MNIST Model..... 3
5 Embedded Wizard Studio..... 6
6 Application Functionality..... 7
7 Accuracy..... 7
8 Implementation Details..... 8
9 Memory Footprint..... 9
10 Extending the Application Example... 11
11 Conclusion..... 12
12 References..... 12

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 present an example solution based on NXP's SDK and the eIQ™ technology.

Handwritten digit recognition with models trained on the MNIST dataset 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 Dataset

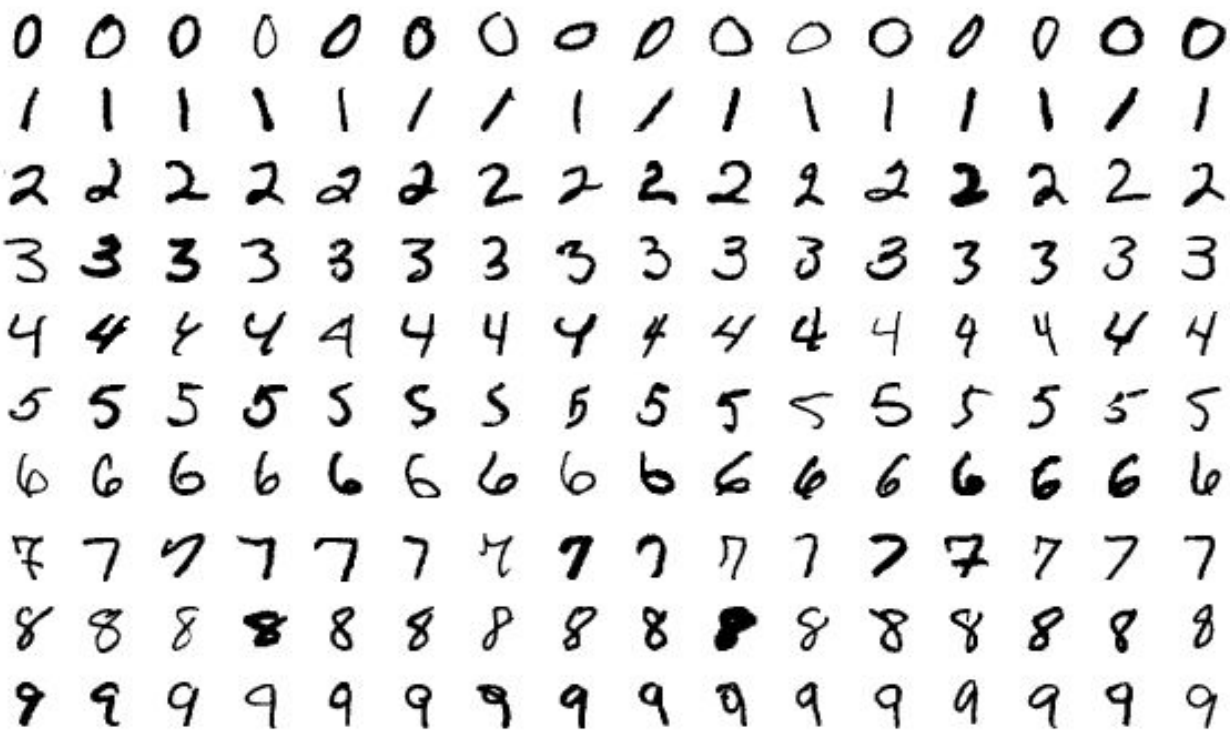

Figure 1. MNIST dataset example (Steppan, 2017)

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.

4 MNIST Model

The model implementation chosen for this example is available on GitHub (TensorFlow, 2019) as one of the official TensorFlow models under the Apache 2.0 license. It is written in python and uses the Keras library and tf.data, tf.estimator.Estimator, and tf.layers APIs. The script builds a convolutional neural network that can achieve over 99 % accuracy on the test set examples from the MNIST dataset. The model definition and the corresponding TensorFlow Lite graph can be seen in Figure 2 and Figure 3.

The Figure 2 graph was generated with Netron (Roeder, 2019), 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.

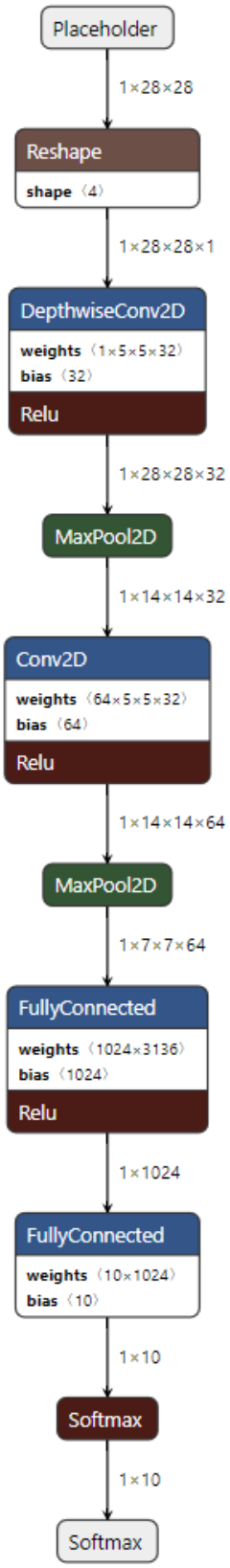

Figure 2. Model visualization in netron (TensorFlow Lite)

```
return tf.keras.Sequential(
    [
        l.Reshape(
            target_shape=input_shape,
            input_shape=(28 * 28,)),
        l.Conv2D(
            32,
            5,
            padding='same',
            data_format=data_format,
            activation=tf.nn.relu),
        max_pool,
        l.Conv2D(
            64,
            5,
            padding='same',
            data_format=data_format,
            activation=tf.nn.relu),
        max_pool,
        l.Flatten(),
        l.Dense(1024, activation=tf.nn.relu),
        l.Dropout(0.4),
        l.Dense(10)
    ])
```

Figure 3. Keras model definition

The trained model was converted to TensorFlow Lite using tflite_convert. For compatibility purposes with the current (August 2019) version (1.11) of the TensorFlow Lite library used in NXP's SDK, the 1.13.2 version of TensorFlow was used for training and converting the model.

Information about how to download, install and use the tflite_convert utility can be found in the eIQ TensorFlow Lite Library User's Guide available in the documentation bundle for the MCUXpresso SDK (NXP, 2019). More details about the utility itself can also be found on the TensorFlow documentation website (TensorFlow, 2019)

```
tflite_convert
 --saved_model_dir=<saved_model_dir_path>
 --output_file=converted_model.tflite
 --input_shape=1,28,28
 --input_array=Placeholder
 --output_array=Softmax
```

```
--inference_type=FLOAT
--input_data_type=FLOAT
--post_training_quantize
--target_ops TFLITE_BUILTINS
```

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

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, 2019) 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 and IAR 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.

The IDE offers a wide variety of GUI objects and tools, including buttons, touch sensitive 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++, some header files needed to have their contents surrounded by:

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

Although in nearly all cases this had already been done by the project generator itself. Additionally, most of the source and header files had to be moved to a new embeddedwizard folder in the middleware folder in the SDK and a few were moved into the source folder as well. Next, new include paths had to be added to reflect these changes. Lastly, some of the device-specific files like board.c were generated with slight differences between the SDK and the Embedded Wizard projects and had to be compared and properly merged together.

6 Application Functionality

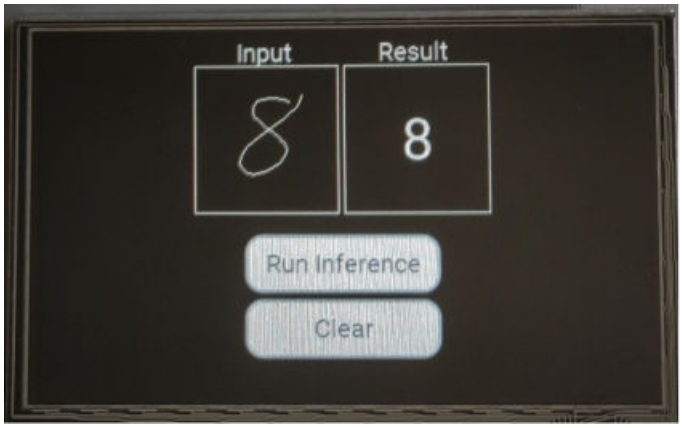

Figure 4. Example inference test

The application is controlled through a GUI displayed on a touch sensitive LCD. The GUI, as shown in Figure 4, 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 5. 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 5. However, mainland European countries, for example, tend to write several of the numbers differently, as apparent in Figure 6, and these styles can lead to wrong predictions.

Figure 6. 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 datasets are taken. Even though the model can achieve over 99 % accuracy on the training and testing datasets, 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. It highlights the importance of training production models on real production data. In order to achieve better results, a new dataset 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 afterwards, 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 dataset. Transfer learning produces very accurate models that are trained faster and require smaller datasets than regular training would need. The NXP Community website contains a walk-through of using the transfer learning technique (Huereca, 2019) to retrain a model from classifying general categories of images to recognizing a small set of flowers.

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. 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 4 times would distort the input too much.

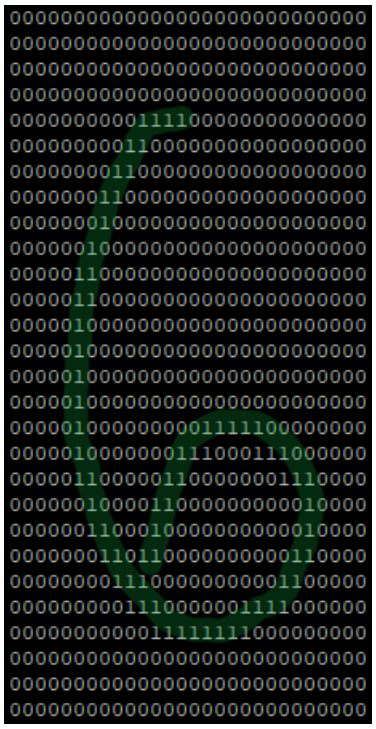

Figure 7. Preprocessed input logging representation (white pixels printed as “1”, black pixels as “0”)

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 3x3 square, 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. Figure 7 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 "middleware/embeddedwizard/Application.c" and then finished in "source/mnist.cpp". The inference and results are handled in "source/mnist.cpp" as well. In "middleware/embeddedwizard/CustomConstant.c" the main and background color constants are defined.

9 Memory Footprint

MCUXpresso and IAR Workbench offer different details when used to inspect memory footprint of an application. This chapter presents the results of both approaches. However, since IAR provides information even about libraries and objects files not relevant to this application note, only selected parts of the memory log are listed here.

Table 1. IAR memory map.

iar\flexspi_nor_sdram_debug\obj	RO Code	RO Data	RW Data
Application.o	2560	448	-
Core.o	24538	2404	-
CustomConstants.o		8	-
DeviceDriver.o	28	-	-
Effects.o	74	120	-
Graphics.o	2234	540	-
Resources.o	488	44030	-
Views.o	12728	644	-
WidgetSet.o	4452	7260	-
bitmap_helpers.o	36	4	-
board.o	736	-	-
clock_config.o	1720	124	-
evkmimxrt1060_flexspi_nor_config.o	512	-	-
evkmimxrt1060_sdram_ini_dcd.o	1072	-	-
ew_bsp_clock.o	676	104	4
ew_bsp_display.o	752	148	4
ew_bsp_event.o	44	-	-
ew_bsp_graphics.o	616	-	58
ew_bsp_inout.o	236	172	4
ew_bsp_serial.o	4	-	-
ew_bsp_system.o	30	-	-

Table continues on the next page...

Table 1. IAR memory map. (continued)

ew_bsp_touch.o	152	36	68
ewextgfx.o	1752	377	9
ewextrte.o	160	24	4
ewmain.o	1220	2012	16
fsl_assert.o	24	44	-
fsl_clock.o	1728	112	8
fsl_debug_console.o	316	180	32
fsl_elcdif.o	414	346	40
fsl_flexspi_nor_boot.o	0	48	-
fsl_ft5406_rt.o	282	120	-
fsl_gpio.o	420	292	-
fsl_lpi2c.o	1724	328	48
fsl_lpuart.o	1008	424	40
fsl_pxp.o	584	238	-
fsl_snvs_hp.o	764	160	-
image.o	494	-	-
lpuart_adapter.o	348	256	-
mnist.o	14532	3284618	258
pin_mux.o	3860	92	-
retarget.o	14	-	-
serial_manager.o	404	156	-
serial_port_uart.o	208	132	-
startup_MIMXRT1062.o	1500	-	-
system_MIMXRT1062.o	330	1	4
tlsf.o	2196	1352	-
Total	**86386**	**3348938**	**597**

Table 2. IAR's information about the relevant static libraries and the Grand Total of the whole application

	RO Code	RO Data	RW Data
libewgfx-m7-iar.a	59838	11019	440
libewrte-m7-iar.a	9748	640	136
libtensorflow-lite.a	388876	49180	3353
Grand Total (whole application)	577990	3413900	8460887

Table 3. MCUXpresso's memory region summary

Memory Region	Used Size	Region Size	%age Used
BOARD_FLASH	4236600 B	8 MB	50.50 %
BOARD_SDRAM	8399652 B	30 MB	26.70 %
BOARD_SDRAM_NONCACHEABLE	0 GB	2 MB	0.00 %
SRAM_DTC	64 KB	128 KB	50.00 %
SRAM_ITC	0 GB	128 KB	0.00 %
SRAM_OC	0 GB	768 KB	0.00 %

NOTE

The SDRAM stores intermediate products of the model's layers, like convolution results. The 8 MB in Table 3 shows the total amount of heap. The actual heap use can be calculated. For example, monitoring it through custom malloc/free functions. For this application, the heap use is 338 KB.

Table 4. MCUXpresso size output

text	data	bss	dec
4232488	4112	8461076	12697676

10 Extending the Application Example

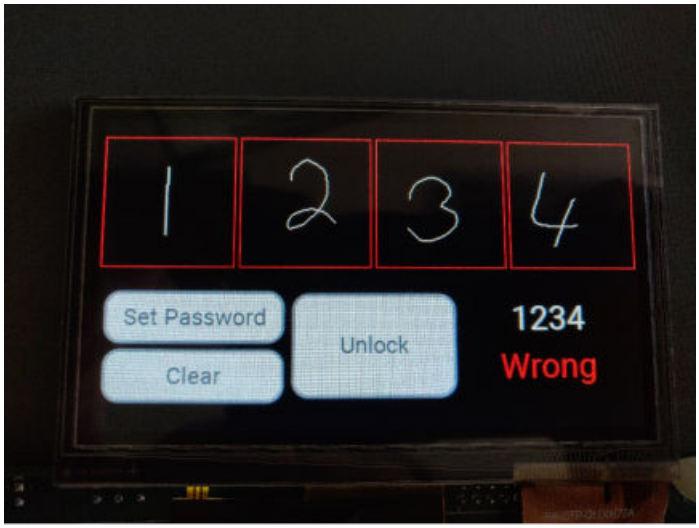

Figure 8. Wrong pin

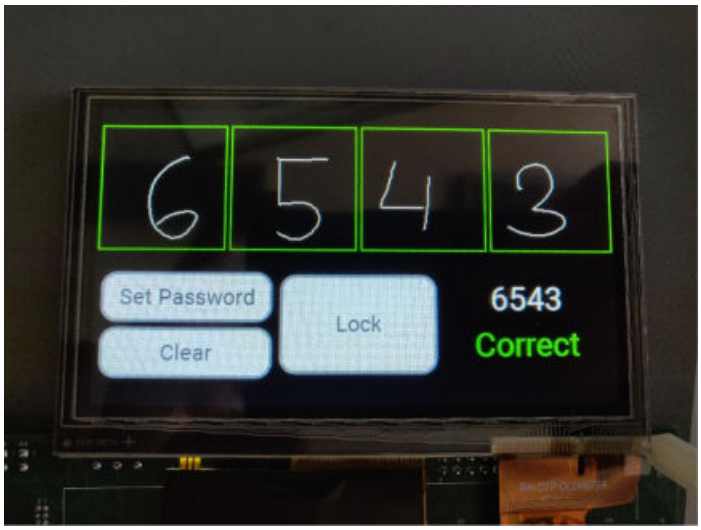

Figure 9. Correct pin

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 8 and Figure 9 was implemented.

To achieve this, the whole input area is cloned three times and the processing is 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.

11 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 EVK-RT1060. 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.

12 References

- LeCun, Y. (2019). *LeNet-5, convolutional neural networks*. Retrieved from Yann LeCun: http://yann.lecun.com/exdb/lenet/
- LeCun, Y. (2019). *The Mnist Database*. Retrieved from Yann LeCun: http://yann.lecun.com/exdb/mnist/
- Steppan, J. (2017, December 14). *Sample images from MNIST test dataset*. Retrieved from wikimedia: https://upload.wikimedia.org/wikipedia/commons/2/27/MnistExamples.png
- TARA Systems GmbH. (2019). Retrieved from Embedded Wizard: https://www.embedded-wizard.de/
- TensorFlow. (2019). TensorFlow Image Classification Repository. Retrieved from GitHub: https://github.com/tensorflow/models/tree/master/official/vision/image_classification
- Wagner, D. J. (2011, April 27). *Numbers and Counting: American vs. French*. Retrieved from ielanguages: 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. (2019). Netron. Retrieved from GitHub: https://github.com/lutzroeder/netron
- NXP. (2019). MCUXpresso Software Development Kit (SDK). Retrieved from NXP: https://mcuxpresso.nxp.com/
- TensorFlow. (2019). TensorFlow Lite converter. Retrieved from TensorFlow: https://www.tensorflow.org/lite/convert
- Weigert, J., Nugent, T., & Moolenaar, B. (2019). xxd(1) - Linux man page. Retrieved from die.net: https://linux.die.net/man/1/xxd
- Huereca, A. (2019). eIQ Transfer Learning Lab with i.MX RT. Retrieved from NXP Community: https://community.nxp.com/docs/DOC-343827

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 20 April 2020

Document identifier: AN12603