

Car License Plate Recognition Based on Keras

Example Code Introduction for 32-bit NuMicro® Family M480 Series

Information

Application	This document describes how to program Keras weight for deep learning technology to develop car plate recognition, and to help users to implement the car plate recognition on the NuMicro® M480 series microcontroller.
BSP Version	M480 BSP CMSIS V3.05.001
Hardware	NuMaker-IoT-M487 Ver1.2

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1 Overview

The global market of machine learning is flourishing recently with the advancement of technology. Machine learning refers to an accumulative and autonomous behavior enhancement from a machine through a series of learning process. The learning process is to give training data to mathematical data models, and it could be categorized into supervised, unsupervised and reinforcement learning.

The idea of machine learning can be realized in almost every field; social media features, product recommendations on the Internet, image recognition, and language translation are all examples of machine learning. According to Fortune Business Insights, the revenue of global machine learning market is predicted to grow dramatically to hit USD 117.19 billion by the end of 2027, with a CAGR of 39.2% compared to the revenue of USD 8.43 billion in 2019.

With DNN (Deep Neural Networks) and CNN (Convolution Neural Networks) that support machine learning networks, the NuMicro[®] M487 Ethernet series from Nuvoton, a high performance and low power microcontroller, is suitable to be used in related applications. The M487, with operating voltage from 1.8 to 3.6V, is powered by Arm[®] Cortex[®]-M4 core with DSP extension, and can run up to 192 MHz with 175 μ A/MHz dynamic low power consumption. The M487 has up to 2.5 MB Flash memory and 160 KB embedded SRAM, which includes 32 KB cache to speed up external SPI Flash code execution. Furthermore, it is equipped with 10/100 Mbps Ethernet MAC with RMII and hardware cryptography engine.

One use case of M487 is vehicle license plate recognition. A M480 platform (with M487 on it) could recognize any vehicle license plate by using its learning neural network algorithms. A CMOS sensor is required to capture the plate image and it takes around 200 ms to identify the image. The M487 supports the image resolution of QVGA 320 x 240.

This sample code captures car license plate, using machine learning neural network algorithms for identification. The Arm[®] Cortex[®]-M4 core supports the DSP instruction to speed up algorithms. The information can be transmitted to a peripheral such as UART on edge device.

1.1 System Overview

This demo captures license plate, using machine learning neural network algorithms for identification. The Arm® Cortex®-M4 core supports the DSP instruction to speed up algorithms. The system architecture is shown in Figure 1-1.

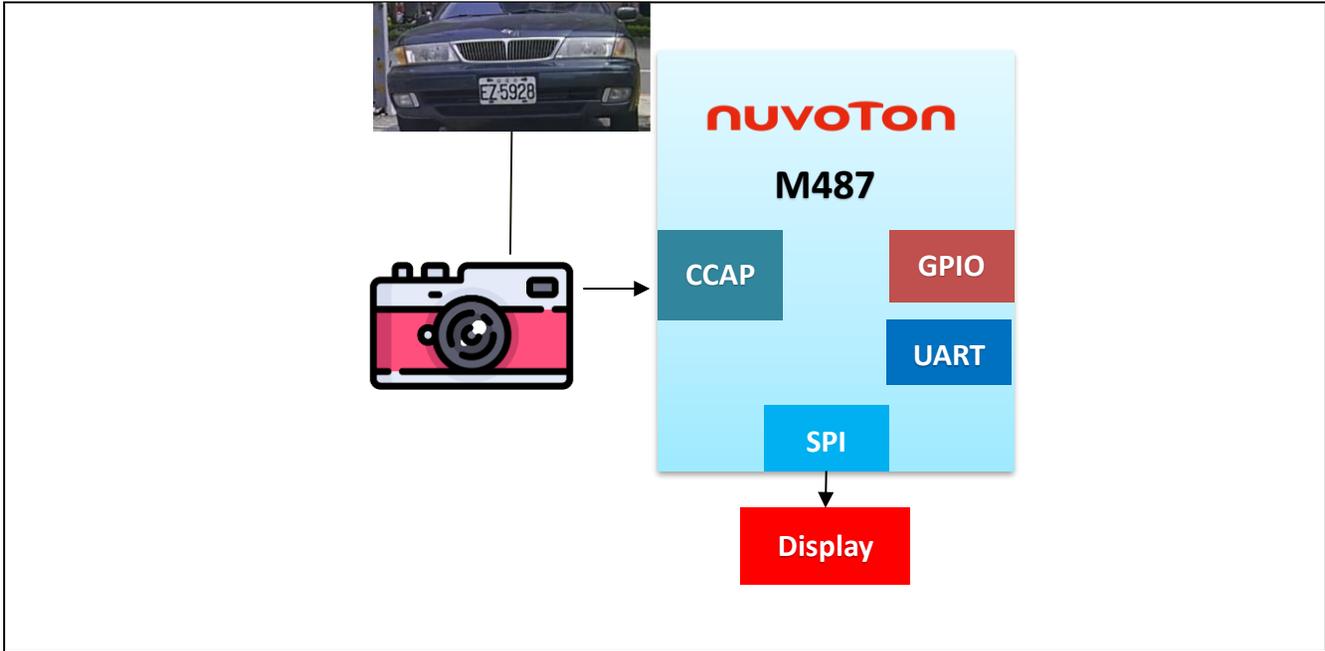


Figure 1-1 System Architecture

1.2 Features

- Hardware/Software driven CMOS sensor
- QVGA: 320*240
- Supports deep neural network and convolutional neural network
- Provides multiple image processing algorithms

1.3 Deep Learning Neural Networks (DNN) Introduction

Machine learning is a branch of Artificial Intelligence (AI). The operation process is to use an algorithm to train a large amount of data. After the training is completed, a model will be generated. When there is new data in the future, user can predict the new data using the training model. Machine learning applications are quite extensive, such as recommendation engines, targeted advertising, demand forecasting, spam filtering, medical diagnostics, natural languages processing, search engines, fraud detection, securities analysis, visual identification, speech recognition, handwriting recognition, and more.

Deep learning is a branch of machine learning. It is the fastest growing field of artificial intelligence. There are several deep learning frameworks, such as Deep Neural Networks (DNN) and Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), etc.. Actually the answer that the user should use what kind of architectures is not certain. It depends on the size, nature, acceptable calculating time, the urgency of learning, and what you want to do with this data. Deep learning is particularly effective in applications such as visual recognition, speech recognition, natural language processing, and biomedical applications.

Deep learning lets the machine simulate the working mode of the human brain, and thus has the same learning ability as human beings. However, the human neural network is too complicated to be simulated. The neurons are divided into multiple levels to simulate the neural network. The neural network usually has an input layer, an output layer and a number of hidden layers that can be trained. More than 2 hidden layers can be called Deep Learning.

If users do not have a basic understanding of deep learning, the course of deep learning can be used to get the most out of this document.

1.3.1 Deep Learning Neural Networks (DNN)

Deep Neural Networks (DNN) is the most basic discriminant model for deep learning. It can be trained by the backpropagation algorithm. The weights can be updated iteratively by the gradient descent method:

$$\Delta w_{ij}(t + 1) = \Delta w_{ij}(t) + \eta \frac{\partial C}{\partial w_{ij}}$$

η is the Learning Rate, C is the Loss Function and the choice of the function is related to the Active Function. For this document as example, in order to solve the problem of Multi-Class Classification supervised learning, choose Rectified Linear Unit (ReLU) as the activation function, and use Cross Entropy as the loss function. The definition of cross entropy is as follows:

$$C = - \sum_j d_j \log(p_j)$$

d_j represents the target probability of output unit j , p_j represents the probability output unit j after applying the activation function; and the Softmax Function of the output layer is defined as follows:

$$p_j = \frac{\exp(x_i)}{\sum_k \exp(x_k)}$$

p_j represents the probability of category j . x_i and x_k are inputs to units i and k , respectively.

The network architecture of the deep neural networks is shown in Figure 1-2.

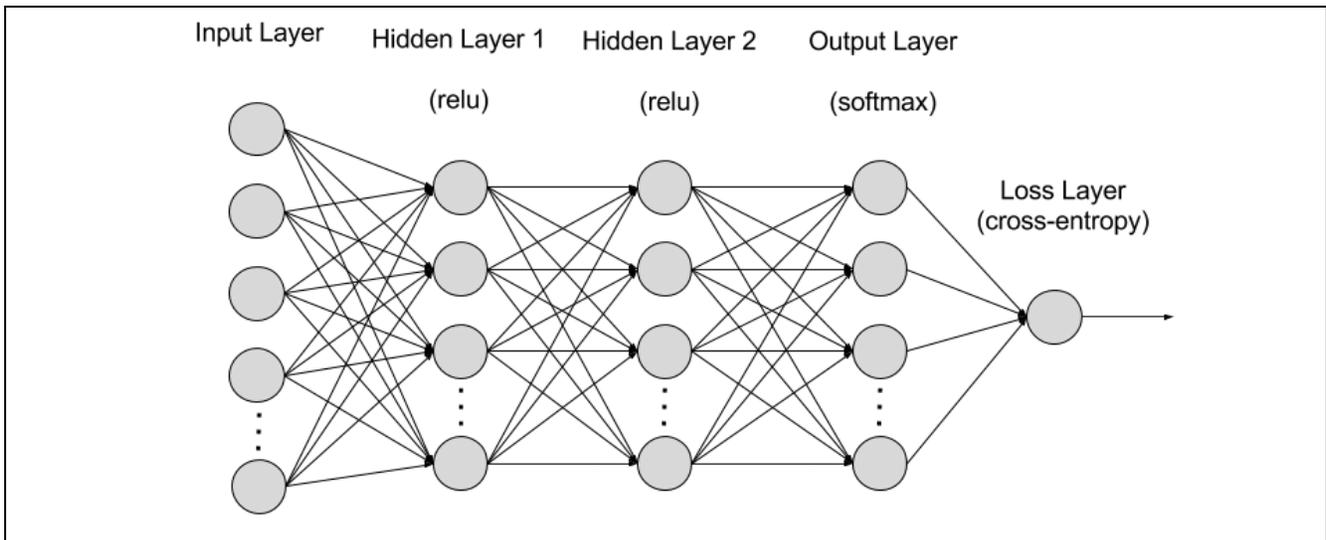


Figure 1-2 Architecture of Deep Neural Networks (DNN)

1.4 TensorFlow

TensorFlow is an open source code library provided by Google. Google has many products that use TensorFlow technology to develop deep learning and machine learning functions (Figure 1-3), such as Gmail filtering spam, Google voice recognition, Google image recognition, Google Translate, etc. The introduction to deep learning has described the core of deep learning and simulate neural networks with tensor (matrix) operations. Accordingly, the main design of TensorFlow is to maximize the performance of matrix operations and used to develop on different platforms.

1.4.1 TensorFlow Architecture

The TensorFlow architecture is shown below.

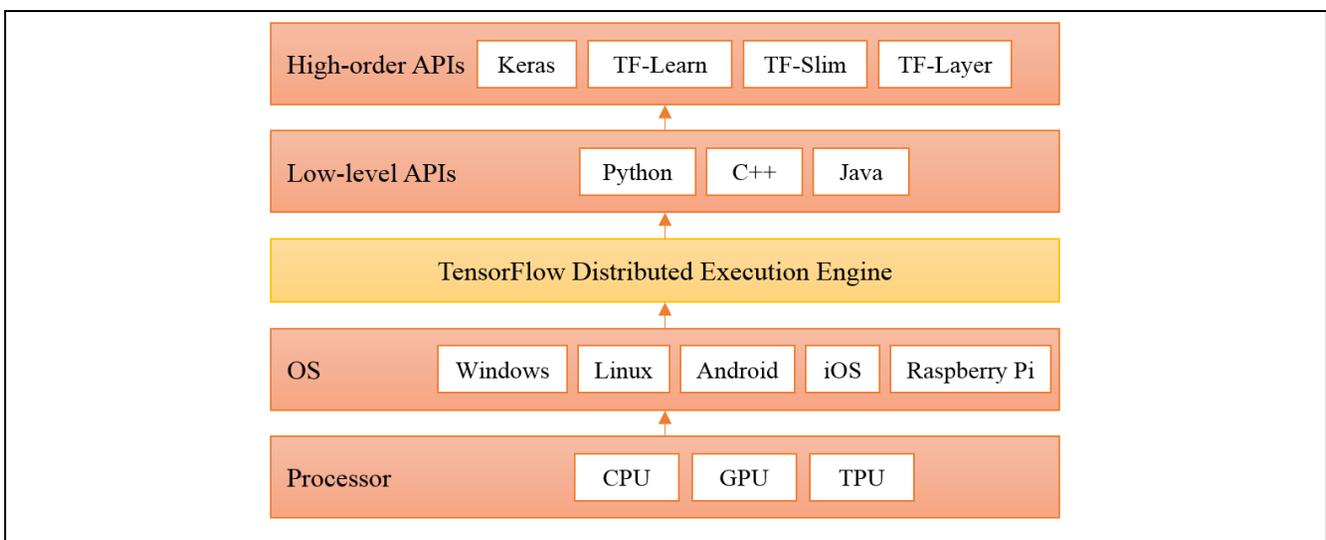


Figure 1-3 TensorFlow Architecture Diagram

The following describes the details of the above architecture diagram, starting from the bottom:

- **Processor:** TensorFlow can be executed on CPU, GPU, and TPU.

- **CPU:** Each computer has a central processing unit (CPU) that can execute TensorFlow. It is enough to use CPU for this speech recognition system.
- **GPU:** Graphics processor with thousands of small and high-efficiency cores that harness the power of parallel computing.
- **TPU:** The Tensor Processing Unit is a proprietary chip developed by Google's Artificial Intelligence and has better execution capabilities than the GPU.

➤ **Platform**

TensorFlow is a cross-platform capability that can be implemented on current mainstream platforms. This document uses the Windows 10 operating system.

➤ **TensorFlow Distributed Execution Engine**

In deep learning, the most time-consuming is the training of the model, especially the large-scale deep learning model, which must be trained with a large amount of data. TensorFlow has the ability of distributed computing, which can perform model training on hundreds of machines at the same time, greatly shorten the time of model training.

➤ **Low-level APIs**

TensorFlow is available in a variety of programming languages and Python has a concise, easy-to-learn, high-productivity, object-oriented, and functional dynamic language that is widely used. The deep learning code of this document is also developed by Python.

➤ **High-order APIs**

TensorFlow is a relatively low-level deep learning APIs. When designing a model, users must design the underlying operations such as tensor product and convolution. Therefore, this document is matched with the high-order API – Keras, which makes developers use more concise and readable codes to construct a variety of complex deep learning models.

1.4.2 Car License Plate Pre-Trained Model

The parameter weights and biases of deep learning models are usually trained using floating-point numbers, but the floating-point numbers can be converted into integers during the micro-controller prediction process without affecting the prediction.

car_weright.h is shown as Figure 1-4.

The parameters of the trained neural network model are quantized into integers, and then the parameters are put into the code of the M480 series microcontrollers for predictive identification.

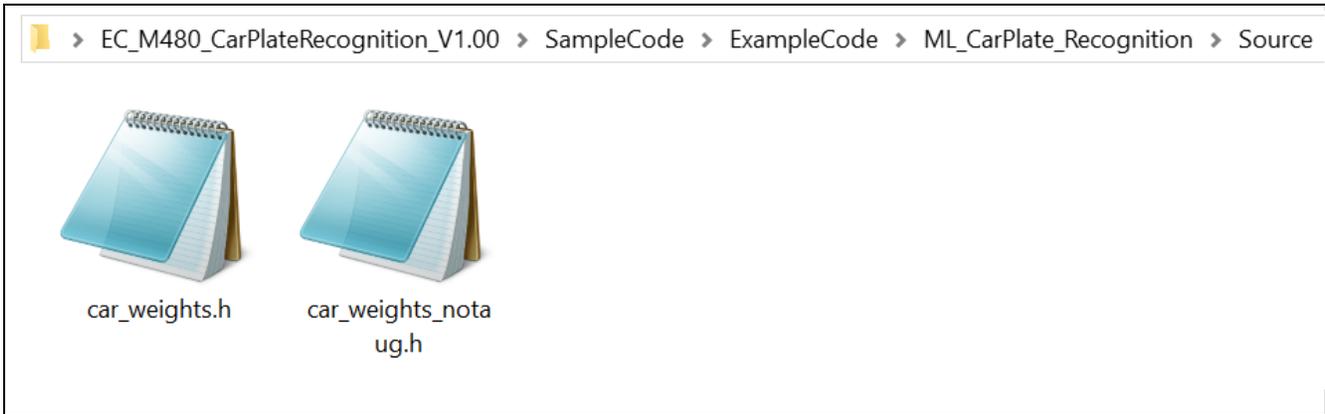


Figure 1-4 car_weights.h

The following is the car license plate recognition model being trained. The program will call this parameter according to the various network layers required by the nnom library to achieve the result of the inference.

```
#include "nnom.h"

/* Neural Network */

#define CONV2D_KERNEL_0 {18, ..., 7, 22}
#define CONV2D_KERNEL_0_SHIFT (4)
#define CONV2D_BIAS_0 {44, 7, 50, 41, -60, 63, -49, -11, 80, 77, 35, -81, -25, -86, 94, -75}
#define CONV2D_BIAS_0_SHIFT (6)
#define CONV2D_1_KERNEL_0 {1, -4, 8, 1, ..., -37, 19, -2, 5, 19, -5, 19}
#define CONV2D_1_KERNEL_0_SHIFT (9)
#define CONV2D_1_BIAS_0 {-47, -80, -6, -12, -64, -64, -22, -14, -3, 19, -35, -28, -18, -7, -61, -60, -46, -29, -69, -3, -56, 1, -27, -9, -37, -66, -48, -54, -24, -6, -50, -55}
#define CONV2D_1_BIAS_0_SHIFT (7)
#define CONV2D_2_KERNEL_0 {-7, -27, -4, 11, ..., -17, -28, 3, -8, 1, 7, -25, -2}
#define CONV2D_2_KERNEL_0_SHIFT (9)
#define CONV2D_2_BIAS_0 {-1, -10, -9, -19, -10, 75, -2, 3, -13, 9, 5, -20, -20, 10, -20, -5, -19, -1, -5, 17, -7, -2, -18, -8, 13, 14, 6, 9, -20, -8, -14, -27, -15, -9, -19, -16, -6, -11, 16, 3, -9, 1, -16, -7, -13, -37, 22, -17, -7, -19, 43, -15, -7, 21, -26, 8, -10, 1, -15, 28, 34, -3, -24, -8}
#define CONV2D_2_BIAS_0_SHIFT (5)
#define CONV2D_3_KERNEL_0 {-11, -20, -19, 0, ..., -8, -9, -11, 3, -9, 5, 9, 3, -18, 16}
#define CONV2D_3_KERNEL_0_SHIFT (8)
#define CONV2D_3_BIAS_0 {-31, -1, -23, 51, -7, -1, -7, -22, -1, 1, 21, 88, 8, 54, 30, 103, -49, -117, 19, 4, 11, -23, -50, 15, 21, 12, -35, 63, -43, -46, -30, -105, -3, -110, 13, 128, 19, -8, 63, 17, 61, -66, 56, 45, 42, 51, 28, -47, -29, -18, 16, -5, 91, -30, -37, 46, -37, -73, 8, 23, 8, -66, 5, 2}
#define CONV2D_3_BIAS_0_SHIFT (7)
```

```
#define DENSE_KERNEL_0 {-16, -18, 2, -9, -15, ..., -7, -38, -5, -21, -12, -2, -10, -19, 69, 1}

#define DENSE_KERNEL_0_SHIFT (8)

#define DENSE_BIAS_0 {-16, -10, -17, -56, -11, -17, -40, -54, -69, -24, -30, -16, 16, -44, -23, -10, -2, -20, -51, -72, 34, -9, 12, -71, -22, 33, -46, -62, 33, -25, 28, -48, -34, 41, 0, 37, 4, -11, 36, -16, -9, 5, -13, 16, -29, -101, -79, 17, -17, 57, -8, -12, 3, -44, 25, -24, 3, -9, -17, 4, -19, -18, 23, -17}

#define DENSE_BIAS_0_SHIFT (11)

#define DENSE_1_KERNEL_0 {-3, -6, 26, -32, -25, ... -56, -26, 27, 25, 4, -53, -46}

#define DENSE_1_KERNEL_0_SHIFT (7)

#define DENSE_1_BIAS_0 {-2, 9, 0, 7, -15, -10, -7, -1, 11, 1, -14, 19, 1, 12, 5, -2, 2, 2, -12, -14, -5, -2, -4, 18, -15, 9, -4, -1, -9, 0, -3, -13, -2, 8}

#define DENSE_1_BIAS_0_SHIFT (7)
```

2 Code Description

This chapter describes how the microcontroller captures the image, and then through the image pre-processing, and finally puts it into the trained model for identification.

2.1 Image Show and Save Function

First, you can define whether to enable two additional functions from main.c, as follows:

```
/* Show image on SPI LCD */
#define SHOW_IMAGE
/* Save image to SD Card */
#define SAVE_IMAGE
```

2.2 Image Capture

This solution uses GPIO software to simulate and capture CCIR601 signals.

```
/* wait Vsync signal */
if(GPIO_GET_INT_FLAG(PE, BIT0)) {
    GPIO_CLR_INT_FLAG(PE, BIT0|BIT1);
    for(i=0; i < IMAGE_HEIGHT; i++) {
        /* wait Hsync signal */
        while(!GPIO_GET_INT_FLAG(PE, BIT1));
        GPIO_CLR_INT_FLAG(PE, BIT1);
        GPIO_CLR_INT_FLAG(PD, BIT8);
        for(j=0; j < IMAGE_WIDTH*PIXEL_BYTES; j++)
        {
            /* latch data when PCLK is high */
            while(!GPIO_GET_INT_FLAG(PD, BIT8));
            GPIO_CLR_INT_FLAG(PD, BIT8);
            /* collect image data from GPIO pin status */
            image[i*IMAGE_WIDTH + j] = (PG->PIN >> 8 );
        }
    }
    GPIO_CLR_INT_FLAG(PE, BIT0);
}
```

If the display function is enabled, you can draw a line on the display to capture the target image, and see the image captured by the CMOS sensor through the display.

```
/* Show image to SPI LCD */
for(i = 0; i < IMAGE_HEIGHT; i++) {
    for(j = 0; j < IMAGE_WIDTH; j++) {
        uint16_t color;
        uint8_t R;
        R = image[i * IMAGE_WIDTH + j];
        color = ((R & 0xF8) << 8) | ((R & 0xFC) << 3) | ((R >> 3));
        LCD_Set_Color(color);
    }
}
```

2.3 Image Recognition

For the defined output category, there are 0~9 and A~Z.

```
/* Output Class */
const uint8_t label_chr[34] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B',
'C', 'D', 'E', 'F', 'G', 'H', 'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'U', 'V',
'W', 'X', 'Y', 'Z'};
```

Classify `nnom_predict` for identification:

```
for(int j = 0 ; j < IMGH ; ++j) {
    for(int i = 0 ; i < IMGW ; ++i ) {
        nnom_input_data[j*IMGW+i] = int8_t (buf_out[c][j*IMGW+i]/2);
    }
}
nnom_predict(model, &predict_label, &predict_prob);
```

You can print the array `label_chr` to show the result.

```
/* identify label function */
predict_label = identify(predict_label, c);
car_plate_result[c] = label_chr[predict_label];
car_plate_prob[c] = predict_prob;
/* show result */
printf("result: %c", label_chr[predict_label]);
printf(" | %.3f ", car_plate_prob[c]);
printf("\r\n");
```

2.4 Save Image

User has to define SAVE_IMAGE first to save the image as a BMP file. The code is as follows.

```
/* Set width and height to save image */
void SaveImage(uint32_t width, uint32_t height, uint8_t *imagedata, uint32_t cnt)
{
    BMP_Header_Init(width, height);
    /* Open bmp file */
    sprintf(Image, "0:\\CMOS%01d.bmp", cnt);
    res = f_open(&file1, Image, FA_CREATE_ALWAYS | FA_WRITE);
    if(res != FR_OK) {
        printf("Open Fingerprint file error!\n");
        return;
    }
    /* Write BMP header*/
    f_write(&file1, &bmp_header, 12, &s1);
    f_lseek(&file1, 14);
    /* Write BMP Infoheader*/
    f_write(&file1, &bmp_Infoheader, 40, &s1);
    /* Write Grayscale Table*/
    f_write(&file1, grayArray, 1024, &s1);
    /* Save image data */
    f_write(&file1, imagedata, width * height, &s1);
    printf("Save Image %d, ", cnt);
    f_close(&file1);
}
```

2.5 Demo Result

Connect the USB port to the PC and open the terminal. When you click the button (SW2 on board), it will print out the result of the identification.

```

COM8 - Tera Term VT
CPU @ 192000000Hz(PLL@ 192000000Hz)
-----+-----
|           Smart image Recognition sample code           |
+-----+-----
Use himax cmos sensor to capture image and uses machine
learning neural network algorithms to identify
Please select capture image method.
mode 0 : fix position, mode 1 : auto detection

File format : BMP file 256 color
result: N | 99.312
result: V | 99.421
result: 5 | 99.184
result: 5 | 99.373
result: 6 | 99.265
result: 0 | 99.372
    
```

Figure 2-1 Demo Result

3 Software and Hardware Requirements

3.1 Software Requirements

- BSP version
 - ◆ M480_BSP_CMSIS_v3.05.001
- IDE version
 - ◆ Keil uVersion 5.26

3.2 Hardware Requirements

- NuMaker-IoT-M487 Board

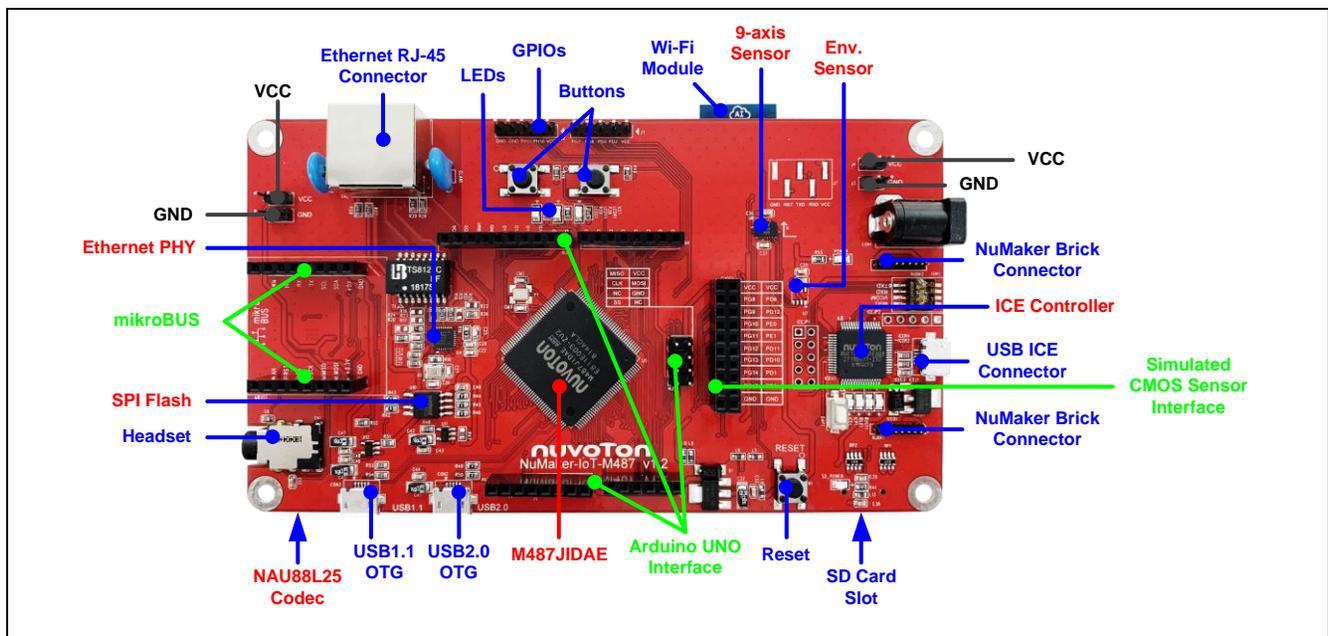


Figure 3-1 NuMaker-IoT-M487 Board

- NuTFT-SPI_320x240 Daughter Board

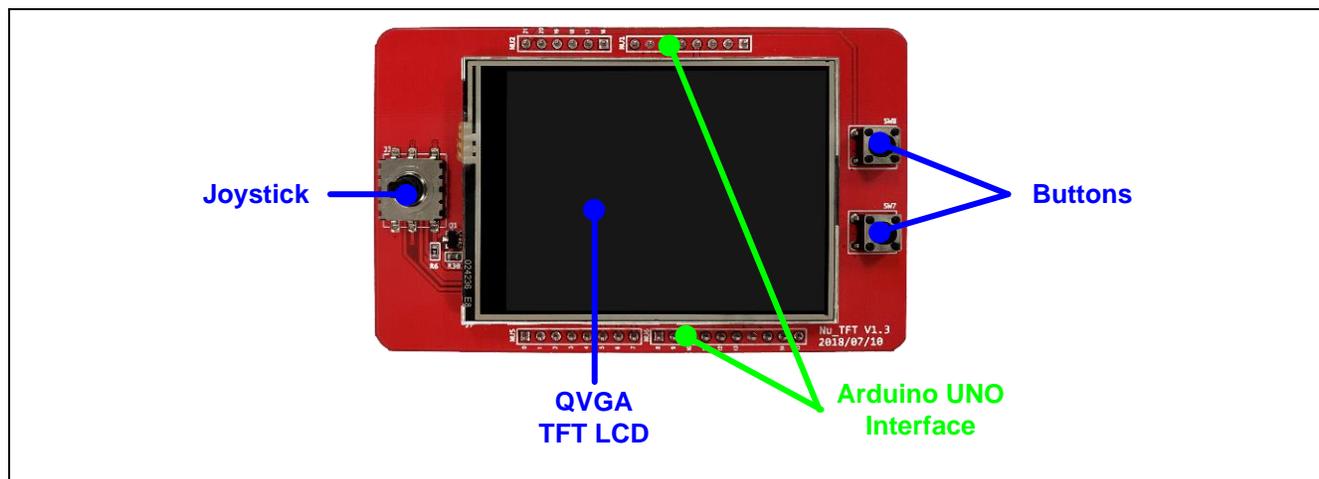


Figure 3-2 NuTFT-SPI_320x240 Daughter Board

- Daughter Board Pin Connect

LCM Interface	LCM Function	M487 Chip	
		GPIO	GPIO Mode
LCM_SPI_SS	LCM SPI chip select	PA.11 (SPI2)	Output
LCM_SPI_CLK	LCM SPI clock	PA.10 (SPI2)	Output
LCM_SPI_MISO	LCM SPI data output	PA.9 (SPI2)	Input
LCM_SPI_MOSI	LCM SPI data input	PA.8 (SPI2)	Output
LCM_DC	LCM command or data	PB.2	Output
LCM_RESET	LCM reset	PB.3	Output
LCM_LED	LCM backlight	PE.5	Output

Table 3-1 Daughter Board Pin Connect

- CMOS image sensor module



Figure 3-3 CMOS Image Sensor Module

- CMOS Pin Connect

CMOS Interface	CMOS Function	M487 Chip	
		GPIO	GPIO Mode
SDA	Serial Data I/O	I2C2_SDA (PD.0)	Input/Output
SCL	I ² C serial clock	I2C2_SCL (PD.1)	Input
MCLK	Master clock input	BPWM0_CH(PD.12)	Input
PCLK	Pixel clock	PE.0	Output
HSYNC	Line valid output	PE.1	Output
VSYNC	Frame valid output	PD.8	Output
D0~D7	Data0~7 output	PG.8~PG.15	Output

Table 3-2 CMOS Pin Connect

4 Directory Information

The directory structure is shown below.

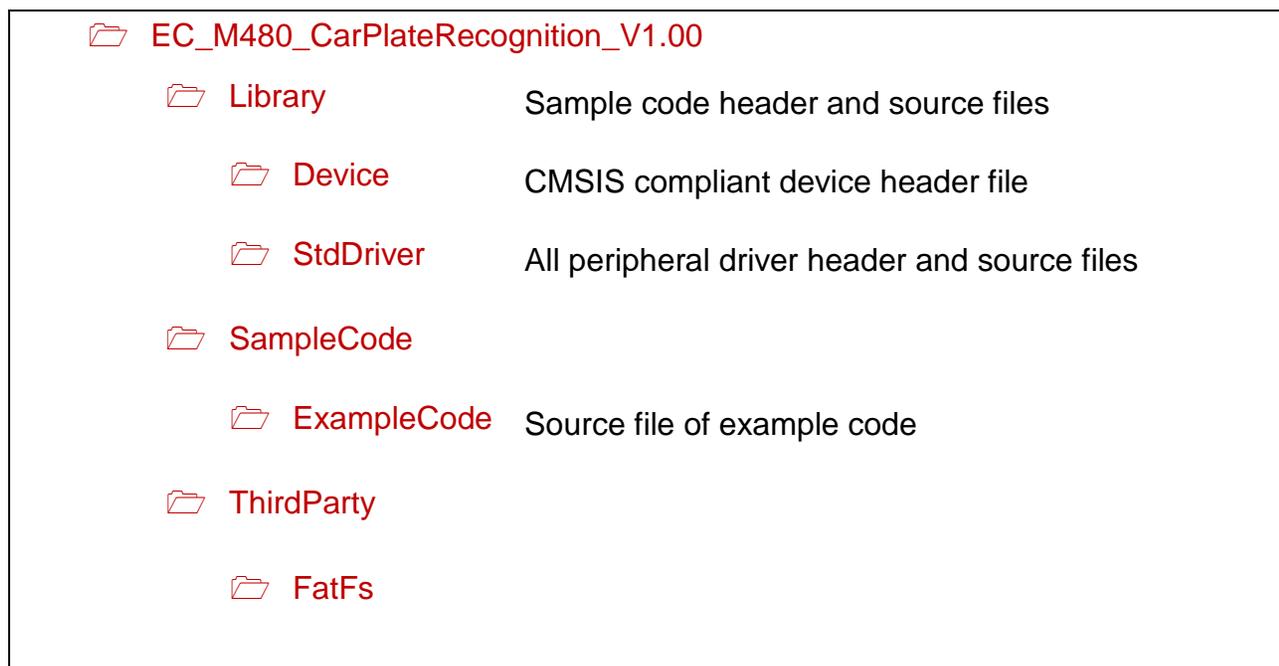


Figure 4-1 Directory Structure

5 Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *ML_CarPlate.uvproj*.
2. Enter Keil compile mode.
 - Build
 - Download
 - Start/Stop debug session
3. Enter debug mode.
 - Run

6 Revision History

Date	Revision	Description
2021.12.15	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

*Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.*