CNN on FRDM\_K64

Contents

[1 Introduction 1](#_Toc61941541)

[2 Experiment 1](#_Toc61941542)

[3 training 5](#_Toc61941543)

# 1 Introduction

Limited by resources, ordinary MCU is difficult to do some complex deep learning. However, although it is difficult, it can still be done. CNN, convolutional neural network, is a kind of deep learning algorithm, which can be used to solve the classification task.

After the implementation of CNN, ordinary MCU can also be used as edge computing device. Next, we introduce how to run CNN on frdm-k64 to recognize handwritten numbers. The size of digital image is 28x28. 28x28 image as input for CNN will output a 1x10 matrix. There are few deep learning libraries written for MCU on the Internet. Even if there are, there will be various problems. NNoM framework is easy to transplant and apply, so we use it

# 2 Experiment

2.1 Required tools: frdm-k64, python 3.7, Pip, IAR, tcp232

2.2 Download the source code of deep learning framework,

https://github.com/majianjia/nnom This is a pure C framework that does not

rely on hardware structure. Transplantation is very convenient

2.3 we select the example ‘bubble’ to add the Inc, port and Src folders in NNoM to the project, as shown in the figure

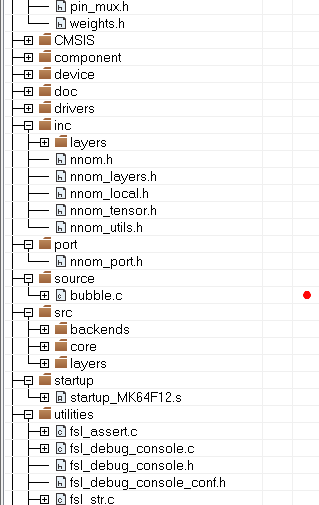


Figure 1

Open the file ‘port.h’ . The definitation of NNOM\_LOG is changed to PRINTF (\_\_ VA\_ ARGS\_\_ ）, Open the ICF file, and change the heap size to 0x5000, define symbol\_\_ size\_ heap\_\_ = 0x5000;

Malloc, which is used in this library, allocates memory from here. If it is small, it can't run the network

2.4 From the download framework, go into ‘mnist-simple/mcu’, which has trained file ‘weights.h’, and randomly generated handwritten image file, ‘image.h’. Add these two files to the project

2.5 Add headfile to ‘bubble.c’

#include "nnom\_port.h"

#include "nnom.h"

#include "weights.h"

#include "image.h"

2.6 Delete the original code, add the following code

nnom\_model\_t \*model;

const char codeLib[] = "@B%8&WM#\*oahkbdpqwmZO0QLCJUYXzcvunxrjft/\\|()1{}[]?-\_+~<>i!lI;:,\"^`'. ";

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Code

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void print\_img(int8\_t \* buf)

{

for(int y = 0; y < 28; y++)

{

for (int x = 0; x < 28; x++)

{

int index = 69 / 127.0 \* (127 - buf[y\*28+x]);

if(index > 69) index =69;

if(index < 0) index = 0;

PRINTF("%c",codeLib[index]);

PRINTF("%c",codeLib[index]);

}

PRINTF("\r\n");

}

}

// Do simple test using image in "image.h" with model created previously.

void mnist(char num)

{

uint32\_t predic\_label;

float prob;

int32\_t index = num;

PRINTF("\nprediction start.. \r\n");

// copy data and do prediction

memcpy(nnom\_input\_data, (int8\_t\*)&img[index][0], 784);

nnom\_predict(model, &predic\_label, &prob);

//print original image to console

print\_img((int8\_t\*)&img[index][0]);

PRINTF("\r\nTruth label: %d\n", label[index]);

PRINTF("\r\nPredicted label: %d\n", predic\_label);

PRINTF("\r\nProbability: %d%%\n", (int)(prob\*100));

}

int main(void)

{

uint8\_t ch;

/\* Board pin, clock, debug console init \*/

BOARD\_InitPins();

BOARD\_BootClockRUN();

BOARD\_InitDebugConsole();

/\* Print a note to terminal \*/

model = nnom\_model\_create();

// dummy run

model\_run(model);

PRINTF("\r\nwhich image to distinguish？ 0-9 \r\n");

for(uint8\_t i=0; i<10; i++)

{

print\_img((int8\_t\*)&img[i][0]);

}

while(1)

{

PRINTF("\r\nwhich image to distinguish？ 0-9 \r\n");

ch = GETCHAR();

if((ch >'9') || ch < '0')

{

continue;

}

PRINTF("\r\n");

mnist(ch-'0');

}

}

An error will be reported when compiling ‘weights.h’, due to lack of few parameters.

In layer [1], layer [4], layer [7], you need to add ‘division (1,1)’ after ‘stride (1,1)’. In this way, the compilation passes.

2.7 As a result, open the serial port software. At the beginning, the terminal will print out a variety of handwritten digital pictures, and then enter a number,

The corresponded picture will be recognized.

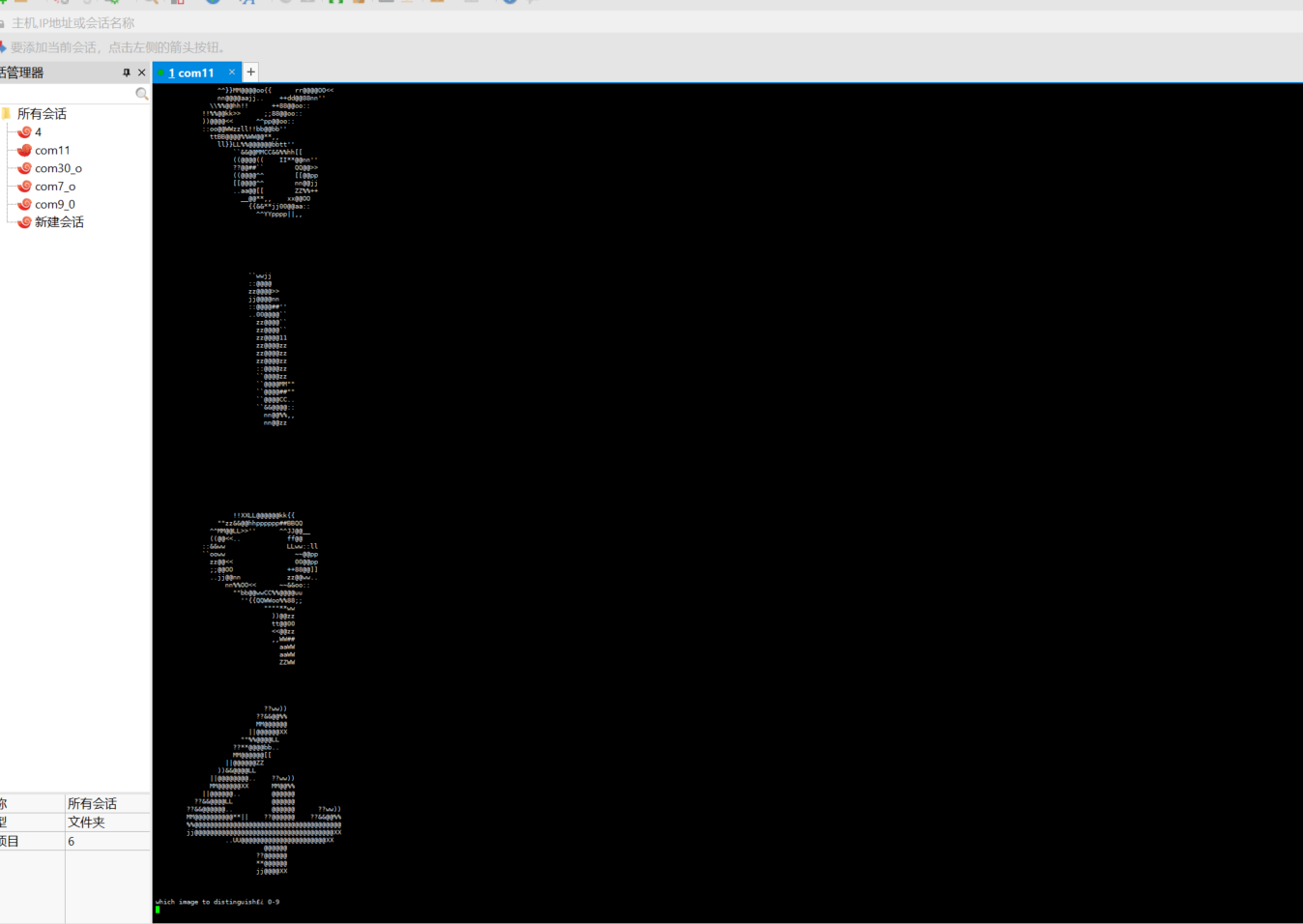


Figure 2

When we input ‘8’, the recognition is the handwriting '9'



Figure 3

The ‘Truth label’ corresponds to IMG9\_LABLE in ‘image.h’ and ‘Predicted label’ is the prediction results

# 3 training

Through the above steps, we have realized a simple handwritten numeral recognition. Next, we will introduce ‘weights.h’. How to generate the weight model here? The image data here are all from MNIST digital set. How can we make a handwritten number for MCU to recognize?

3.1 Under ‘nnom-master\examples\mnist-simple’, there is a ‘mnist\_ simple.py’. You need to run it to generate ‘weights.h’ and ‘image.h’. To run this, you need to install tensorflow, keras and so on. When you run it, you can use pip to install what is missing

The network operation process is as shown in the figure

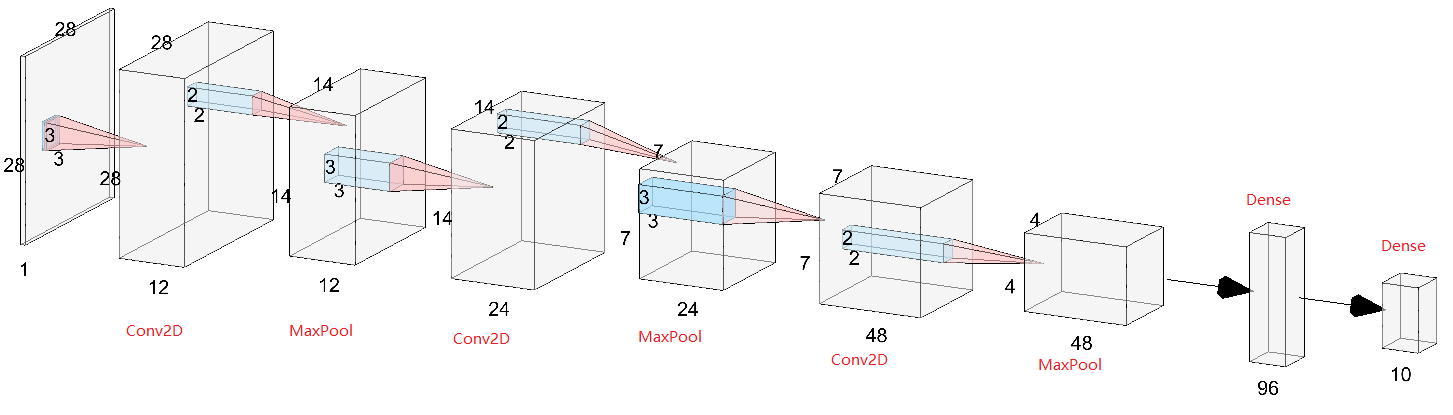


Figure 4

Conv2d-> convolution operation, Maxpool-> pooling.

The meaning of convolution operation is to extract the features of the image. Pooling is a bit like compressing data, which can reduce the running space.

28x28 input and output a 1x10 matrix, representing the possibility of 0-9

3.2 We can use the ‘Paint’ program of WIN to adjust the canvas to 28x28, write numbers on it and save it in PNG format. I wrote a ‘4’



Figure 5

Change the code as following.

nnom\_model\_t \*model;

uint8\_t temp[28\*28]={0};

const char codeLib[] = "@B%8&WM#\*oahkbdpqwmZO0QLCJUYXzcvunxrjft/\\|()1{}[]?-\_+~<>i!lI;:,\"^`'. ";

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Code

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void print\_img(int8\_t \* buf)

{

for(int y = 0; y < 28; y++)

{

for (int x = 0; x < 28; x++)

{

int index = 69 / 127.0 \* (127 - buf[y\*28+x]);

if(index > 69) index =69;

if(index < 0) index = 0;

PRINTF("%c",codeLib[index]);

PRINTF("%c",codeLib[index]);

}

PRINTF("\r\n");

}

}

void mnist\_pic(uint8\_t \*temp)

{

float prob;

uint32\_t predic\_label;

PRINTF("\nprediction start.. \r\n");

// copy data and do prediction

memcpy(nnom\_input\_data, (int8\_t\*)temp, 784);

nnom\_predict(model, &predic\_label, &prob);

//print original image to console

print\_img((int8\_t \*)temp);

PRINTF("\r\nPredicted label: %d\n", predic\_label);

PRINTF("\r\nProbability: %d%%\n", (int)(prob\*100));

}

int main(void)

{

/\* Board pin, clock, debug console init \*/

BOARD\_InitPins();

BOARD\_BootClockRUN();

BOARD\_InitDebugConsole();

/\* Print a note to terminal \*/

model = nnom\_model\_create();

// dummy run

model\_run(model);

while(1)

{

PRINTF("\r\n Send picture by serial\r\n");

DbgConsole\_ReadLine(temp,784);

PRINTF("\r\n Got picture\r\n");

mnist\_pic(temp);

}

}

3.3 Then use pic2mnist.py(see the attachment), run this script with CMD and

enter 'Python pic2mnist.py 1. PNG ', 1. PNG is the image to be parsed, and then ‘content.txt’ will be generated. The file contains the data of the picture. Send the data to the MCU through the serial port. Note that ‘Send as Hex’ should be checked. Similarly, the handwritten picture will be displayed first, and then the picture will be recognized.

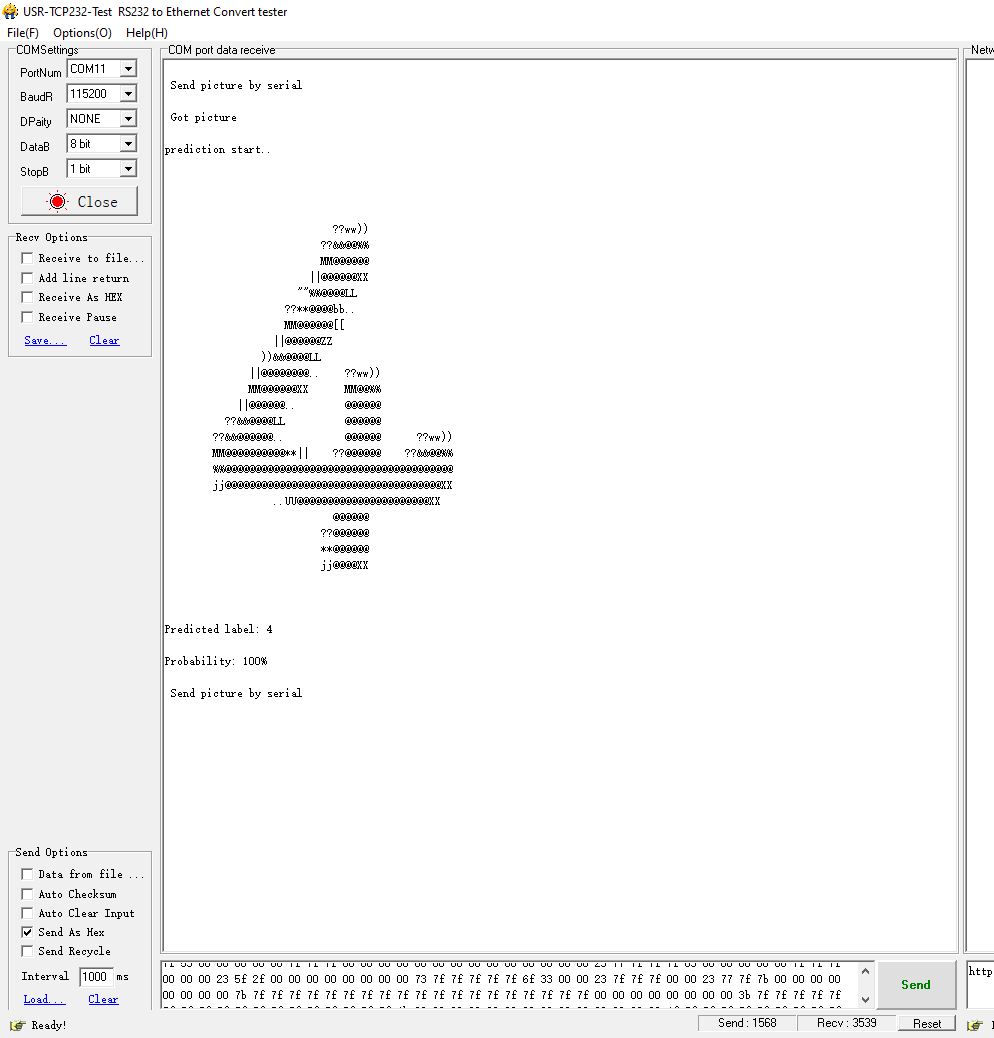


Figure 6

We can see that '4' was identified