基于 STM32H747I-DISCO 的手写数字识别

嵌入式小组 – 2151094 宋正非; 2152482 鄂俊伍; 2152222 邹阳

2024/6/14

1 STM32H747I-DISCO 板子简介与使用

1.1 STM32H747I-DISCO 简介

STM32H747I-DISCO 是 STMicroelectronics（意法半导体）推出的一款高性能微控制器开发板，专为需要高处理能力和复杂功能的嵌入式系统设计。它基于双核架构，包含一个 Cortex-M7 核和一个 Cortex-M4 核。Cortex-M7 核的主频最高可达 480 MHz，而 Cortex-M4 核的主频则为 240 MHz，这种双核设计允许开发者在同一设备上运行实时操作系统和应用程序，从而提高系统的整体性能和灵活性。

STM32H747I-DISCO 开发板提供了丰富的外设和接口，包括多个 USB 端口、CAN 总线、Ethernet（以太网）、SDMMC 接口、SPI、I2C、USART、ADC、DAC 等。这些外设和接口使得 STM32H747 在处理复杂任务时具有很高的适应性和扩展性。开发板还配备了多种存储选项，包括最大 1 MB 的 Flash 存储器和高达 1 MB 的 SRAM，确保了充足的存储空间来支持大型应用和实时数据处理。

此外，STM32H747I-DISCO 还支持丰富的图形处理能力，内置 Chrom-ART 图形加速器和 LCD-TFT 控制器，可以驱动高级图形用户界面（GUI）。这使得它特别适用于需要复杂图形显示的应用，如高级人机界面（HMI）设备、工业自动化控制面板和智能家居系统。

1.2 STM32H747I-DISCO 可用内存

STM32 系列微控制器拥有多样化的内存配置，以满足不同应用场景的需求。其内存架构主要包括 Flash 存储器、SRAM（静态随机存取存储器）和一些特殊用途的存储区域。Flash 存储器用于存储代码和常量数据，容量范围从几十 KB 到几 MB 不等，例如 STM32H7 系列的 Flash 存储器最大可达 2MB。SRAM 用于存储运行时的变量和数据，容量也非常多样，从几 KB 到上百 KB 不等，例如 STM32F7 系列的 SRAM 最大可达 512KB。此外，STM32 微控制器还支持外部存储器接口，如 SDRAM 和 NOR Flash，通过外部存储器接口（FMC、QUADSPI 等），可以进一步扩展存储空间。

除了常规的 Flash 和 SRAM，部分 STM32 系列还包含特定用途的内存区域，例如 EEPROM 模拟区，用于需要非易失性存储的小型数据保存，或专门的缓冲区（如 DTCM、ITCM）来提升特定操作的效率。STM32 系列还配备了灵活的存储管理单元（MMU）和内存保护单元（MPU），以实现更高级的内存管理和安全保护。在开发过程中，STM32CubeMX 等工具可以帮助开发者直观地配置和优化内存使用，结合 STM32CubeIDE 的调试功能，可以有效分析和管理内存使用情况，避免内存泄漏和溢出等常见问题。综合来看，STM32 微控制器提供了丰富的内存资源和灵活的管理机制，使其在各种复杂的嵌入式系统应用中都能表现出色，不论是实时数据处理、高性能计算还是复杂的图形界面处理，都能满足不同层次的需求。

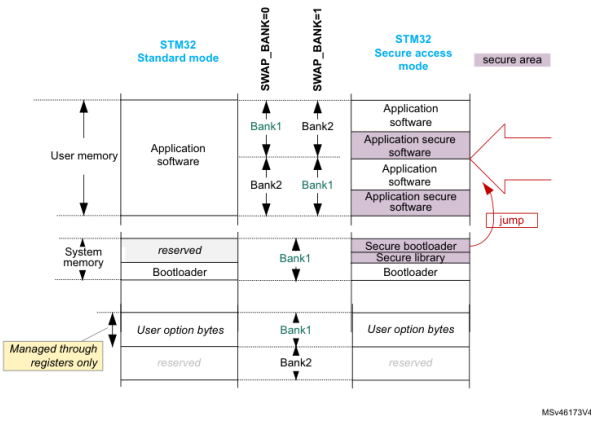


图 1: 内存示意图

1.3 STM32H747I-DISCO 触摸屏调用

STM32H747I-DISCO 开发板上使用触摸屏的方法主要涉及硬件连接和软件配置两个方面。首先，硬件连接方面，需要将触摸屏的信号线正确连接到 STM32H747 的相应引脚，通常包括电源、地、数据线（如 I2C/SPI 接口）和控制线（如中断引脚）。常见的触摸屏控制芯片包括 FT6206、STMPE811 等，需要根据具体型号连接对应的引脚。接下来，软件配置方面，开发者可以使用 STM32CubeMX 工具配置触摸屏相关的外设，如 I2C 或 SPI 接口，用于与触摸控制器通信，同时启用中断以处理触摸事件。在 STM32CubeMX 中配置完成后，生成初始化代码，然后在 STM32CubeIDE 中进行进一步开发。具体实现时，可以利用 STM32 的 HAL 库或 LL 库进行 I2C/SPI 通信，读取触摸控制器的寄存器以获取触摸坐标和触摸事件信息。为了处理和显示触摸数据，通常需要结合图形库，如 TouchGFX 或 emWin，这些库提供了丰富的图形界面组件和触摸事件处理机制。开发者可以在图形库的基础上开发用户界面，并将触摸事件与界面元素交互关联。例如，在 TouchGFX 中，可以配置触摸屏中断处理函数，当触摸事件发生时，读取触摸坐标并传递给图形库进行界面更新。

STM32H747I-DISCO 通过其强大的处理能力和丰富的外设接口，结合图形库的使用，能够实现高效的触摸屏交互，使其在需要图形显示和用户交互的嵌入式系统中表现优异。开发者可以通过合理配置和编程，充分发挥 STM32H747 和触摸屏的功能，实现复杂的触摸界面应用。

1.4 CubeAI 简介

CubeAI 是由 STMicroelectronics 提供的一个强大工具集，旨在将人工智能（AI）和机器学习（ML）算法集成到基于 STM32 微控制器的嵌入式系统中。CubeAI 是一部分 STM32CubeMX 生态系统，通过一个易于使用的图形化界面，帮助开发者将预训练的神经网络模型转换为高效的 C 代码，并在 STM32 微控制器上运行。

CubeAI 的核心功能是模型转换和优化。它支持多种深度学习框架，如 TensorFlow、Keras 和 ONNX，允许开发者将这些框架中训练好的模型导入到 CubeAI 中。CubeAI 会自动进行量化、优化和转换，将浮点运算转换为定点运算，从而在资源有限的嵌入式系统中高效运行。同时，它还进行内存和计算资源的优化，以确保在 STM32 微控制器上的运行效率。使用 CubeAI 的过程通常包括以下几个步骤:

模型导入和转换： 首先，将预训练的神经网络模型导入 CubeAI。这个模型可以是在 PC 上用 TensorFlow、Keras 等训练的深度学习模型。CubeAI 支持导入多种模型格式，包括 HDF5、PB 和 ONNX 等。

优化和量化： 导入模型后，CubeAI 会对模型进行优化和量化处理。优化步骤包括删除冗余的计算节点和压缩模型结构，以减少模型大小和提高推理速度。量化步骤则将浮点模型转换为定点模型，以适应嵌入式系统的计算能力和内存限制。

生成代码： 完成优化和量化后，CubeAI 生成 STM32 微控制器上运行的 C 代码。生成的代码包括神经网络推理函数、初始化代码和必要的库文件。这些代码可以无缝集成到 STM32CubeIDE 项目中。

部署和验证： 将生成的代码部署到 STM32 微控制器上，通过 STM32CubeIDE 进行编译和下载。然后，可以在实际的嵌入式设备上运行和验证神经网络模型的性能和准确性。

2 识别算法构建

识别算法首先在电脑进行训练与检测，现对我们的构建流程进行介绍。

2.1 数据集

MNIST(Modified National Institute of Standards and Technology) 数据集是一个广泛用于计算机视觉领域的手写数字识别任务的数据集，由 60000 个训练样本和 10000 个测试样本组成，包含了 0 至 9 共 10 类手写数字图片，每个样本都是一张 28 * 28 像素的灰度手写数字图片。

MNIST 数据集经常被用作深度学习初学者的入门任务，因为它相对较小，训练相对迅速，同时具有足够的复杂性。

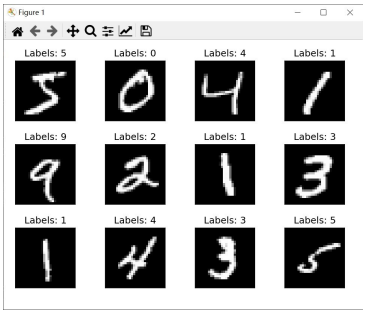


图 2: 数据集样例

2.2 构建流程

定义超参数： 首先，定义超参数，即用于定义模型结构或优化策略的参数；包括每批训练处理的样本数量 batch size、训练数据集的轮次 epoch 以及训练设备的选择。本实验将超参数设定为：batchsize=128, epoch=30, 并选择 device 为 GPU。

处理数据集： 随后，下载数据集，将数据集分为训练集与测试集两类。同时对数据集进行处理转换，将数据集中的样本转为张量形式并进行归一化处理，实现数据增强，从而提高模型的泛化能力和效果，减少模型对某些特征的过度依赖从而避免过拟合。根据每个轮次的样本数量加载数据集并打乱。

构建网络模型： 接着，构建网络模型。典型的 CNN 架构包括卷积层 (Convolutional Layer)、池化层 (Pooling Layer)、全连接层 (Fully Connected Layer) 和激活函数 (Activation Function)。卷积层用于通过卷积操作从输入中提取图像局部特征；激活函数用于引入非线性变换；池化层用于对特征图进行降维，以降低计算量并提高模型的鲁棒性；全连接层用于将特征映射到类别上。本实验网络结构主要由三部分构成，具体如图 3 所示。

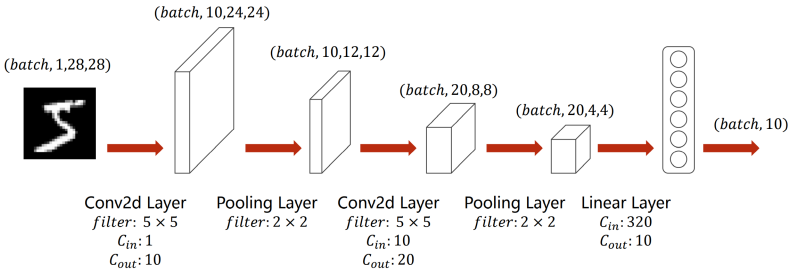


图 3: 网络框架

首先使用一个 5×5 的卷积层，图像通道由 1 变为 10、高宽由 28 变为 24。经过 ReLU 激活后，通过核为 2Œ2 的池化层，使用最大池化法池化后，通道数不变，高宽变为一半。经过上述操作，图像维度由 (batchsize, 1, 28, 28) 变为 (batchsize, 10, 12, 12)。

随后再经过一个 5Œ5 的卷积层，将图像通道由 10 变为 20、高宽由 12 变为 8。经过 ReLU 激活后，通过核为 2Œ2 的池化层，使用最大池化法，使得通道数不变，高宽变为一半。将图像维度变为 (batchsize, 20, 4, 4)。最后再将其展平后进入全连接层，使用线性函数输出为 10 类，对应 0 至 9 共 10 个数字。

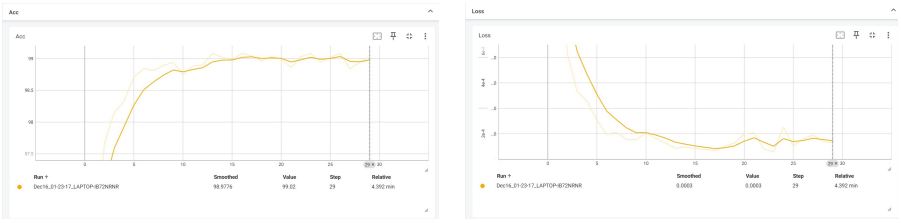
训练过程： 使用训练集对构建好的网络模型进行训练。训练过程主要包括前向传播 (Forward Propagation) 与后向传播 (Backward Propagation)。前向传播为，输入图像通过卷积、池化和全连接层最终得到分类结果；反向传播则根据损失函数 (Loss) 计算梯度，并使用优化算法更新网络参数，最小化损失函数。

其中，损失函数选择交叉熵损失函数 (Cross Entropy Loss) 。其定义为：$L = -\frac{1}{N}\sum_{i\in N}\sum_{c=1}^{M} y_{ic} log(p_{ic})$

优化器 (Optimizer) 选择 Adam(Adaptive Moment Estimation) 优化器，其实现简单，计算高效，能够自动调整学习率 (learning rate)，工作性能较优。其中，本实验将学习率设定为 0.0005。在训练过程中，每一轮训练后通过在测试集上评估得到损失与精确度结果，并使用 TensorBoard 对训练曲线进行绘制，横坐标为训练的轮次，即 30。同时使用 logging 记录训练日志，并以 json 格式输出以记录训练中的模型情况。

模型评估： 图 4 可表示，损失值整体较小，随着训练轮次的增加而减小，在 10 轮前降低速度较快，随后降低速度放缓并趋于 1e-4 左右。准确值反应测试集上正确分类的样本数与所有样本数的比值，其整体较高，在训练前 10 轮增长较为迅速，随后在 99 % 附近震荡。

整体而言，模型的准确度较高、损失值较低，手写数字识别的性能满足预期。



(a) (b)

图 4: Accuracy and Loss

3 算法部署与 CubeAI 应用

3.1 将模型 pt 转成 onnx

深度学习训练出来的模型参数存储在 pt 文件中，将模型部署到板子需要将其转换成 ONNX 或 Keras 或 TFLite。

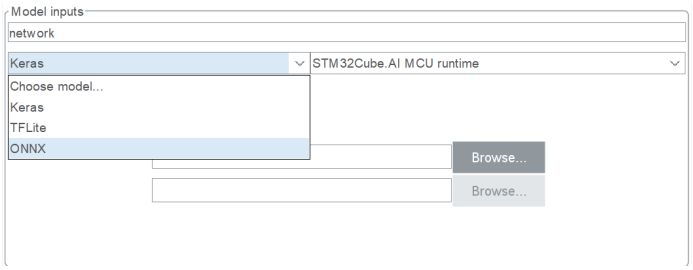


图 5: onnx-choose

我们将模型转换成 ONNX 格式，其是一个开放的神经网络交换格式，用于在不同的深度学习框架之间共享模型。其具有如下特点：（1）互操作性：其可以在多个深度学习框架如 PyTorch、TensorFlow、Caffe2 等之间互相转换和共享。（2）标准化：提供了一个标准格式，使得不同工具和框架可以更容易地集成。（3）广泛支持：许多主流的深度学习框架和硬件供应商都支持 ONNX，提供了广泛的兼容性。

3.2 选择 AI 工具包

创建项目选择能运行神经网络的 MCU，STM32F4 以上系列的都支持部署神经网络。点击 Middleware and Software Packs，选择 X-CUBE-AI，选择最近的 CUBEAI 工具包。不同版本的生成的代码有些会不一样，有的模型老版本可以部署新版本就不行了。

3.3 导入模型

通过分析与压缩得到模型分析后的结果，会有模型的输入和输出以及模型的 Flash 和 RAM。点击 show graph 可以看到网络的模型。确定无误后直接生成代码即可。

3.4 CubeAI 的使用

根据官方提供的示例代码可知，自己需要修改的地方主要有三点。1、定义网络句柄、输入、输出和激活缓冲区数据 buf 和管理输入和输出数据的指针。2、获取和处理数据并进行推理。如果是图像分类任务，图像数据可能是来自摄像头模块获取。其他任务例如运动检测任务，数据可能会来自六轴加速度传感器。3、对

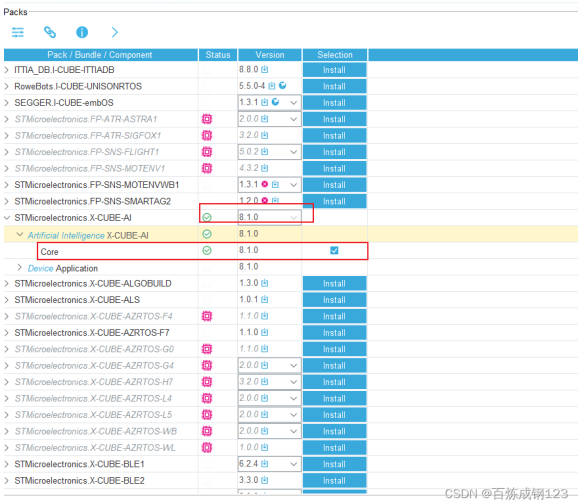


图 6: 选择工具包

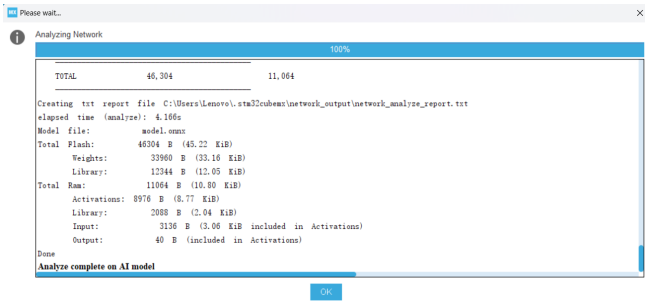


图 7: 导入模型

输出数据进行后处理。对于分类任务，模型输出结果是每个类别的概率，即 1×10 大小的 tensor(如图 6 所示)，我们要输出准确率最高的类别就要写一个求最大值的函数。

4 实现功能

4.1 添加辅助文件

在工程目录下创建新文件夹 BSP，并将固件库中的 BSP 文件夹中的 Components、STM32H747I-DISCO 文件夹、固件库中的 Utilities 文件夹全部拷贝到工程新建的 BSP 中目录中。3) 在工程的 BSP 中创建一个 Inc 文件夹，分别将 $BSP/Components/is42s32800j$ $is42s32800j_conf_template.h$ $BSP/Components/ft6x06$ 中的 $ft6x06_conf_template.h$ $BSP/Components/mt25tl01g$ $mt25tl01g_conf_template.h$ 以及 BSP/STM32H747I-DISCO 文件夹中的 stm32h747idiscoveryconftemplate.h、文件拷贝到 Inc 文件夹中，并重命名为 is42s32800jconf.h，ft6x06conf.h, mt25tl01gconf.h 和 stm32h747idiscoveryconf.h，接着根据需求修改 stm32h747idiscoveryconf.h。最后将上述文件在 Keil 中配置好。

4.2 使用提供好的触摸屏函数

打开 stm32lcd.c，该文件里有官方提供的触摸屏使用函数，可以根据需求直接调用其中的函数来实现功能。

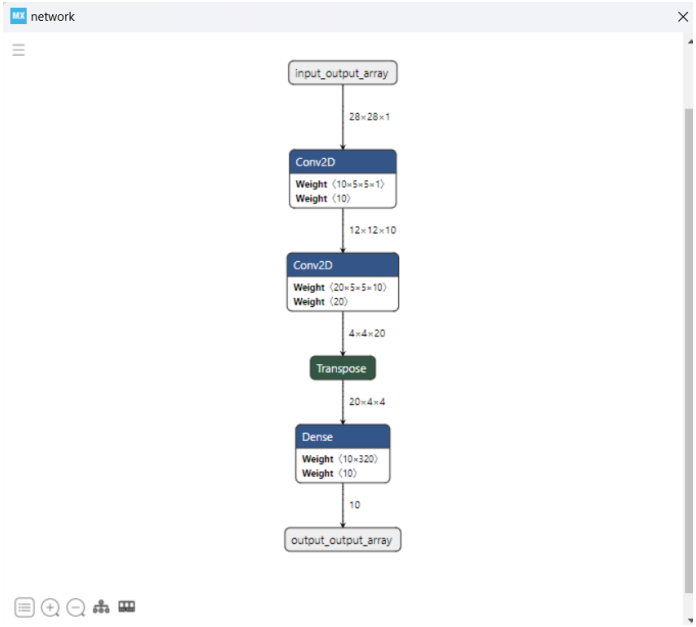


图 8: 模型框架

```
- At application level, once the LCD is initialized, user should call UTIL_LCD_SetFuncDriver()
  API to link board LCD drivers to BASIC GUI LCD drivers.
  User can then call the BASIC GUI services:
      UTIL_LCD_SetFuncDriver()
      UTIL_LCD_SetLayer()
      UTIL_LCD_SetDevice()
      UTIL_LCD_SetTextColor()
      UTIL_LCD_GetTextColor()
      UTIL_LCD_SetBackColor()
      UTIL_LCD_GetBackColor()
      UTIL_LCD_SetFont()
      UTIL_LCD_GetFont()
      UTIL_LCD_Clear)
      UTIL_LCD_ClearStringLine()
      UTIL_LCD_DisplayStringAtLine()
      UTIL_LCD_DisplayStringAt()
      UTIL_LCD_DisplayChar()
      UTIL_LCD_GetPixel()
      UTIL_LCD_SetPixel()
      UTIL_LCD_FillRGBRect()
      UTIL_LCD_DrawHLine()
      UTIL_LCD_DrawVLine()
      UTIL_LCD_DrawBitmap()
      UTIL_LCD_FillRect()
      UTIL_LCD_DrawLine()
      UTIL_LCD_DrawRect()
      UTIL_LCD_DrawCircle()
      UTIL_LCD_DrawPolygon()
      UTIL_LCD_DrawEllipse()
      UTIL_LCD_FillCircle()
      UTIL_LCD_FillPolygon()
      UTIL_LCD_FillEllipse()
```

图 9: 提供好的触摸屏函数

4.3 实现样例

如图 10 所示，是我们开发手写数字识别的实际演示效果。以该图为例，在左侧书写区域中书写数字“4”，能正确识别并在板子上输出预测结果。除此之外，我们对其余数字都进行了实际测试，效果优异，详情可见附件视频。

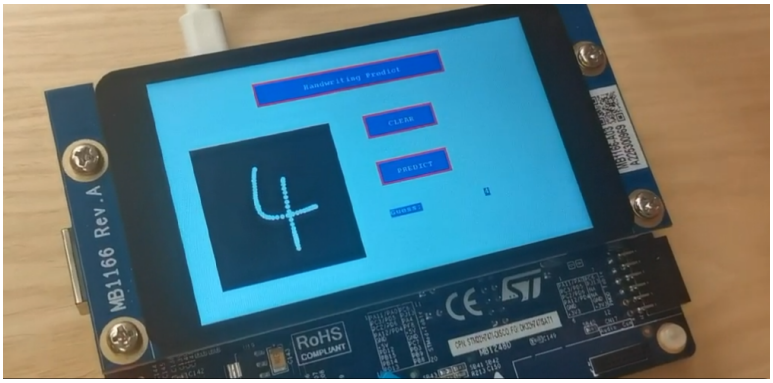


图 10: 实际效果

5 问题与解决

5.1 main 文件重名且内存不足

起初，我们自行编写了基于 C 语言的识别算法代码，并部署到板子，但发现会出现如图所示的问题。

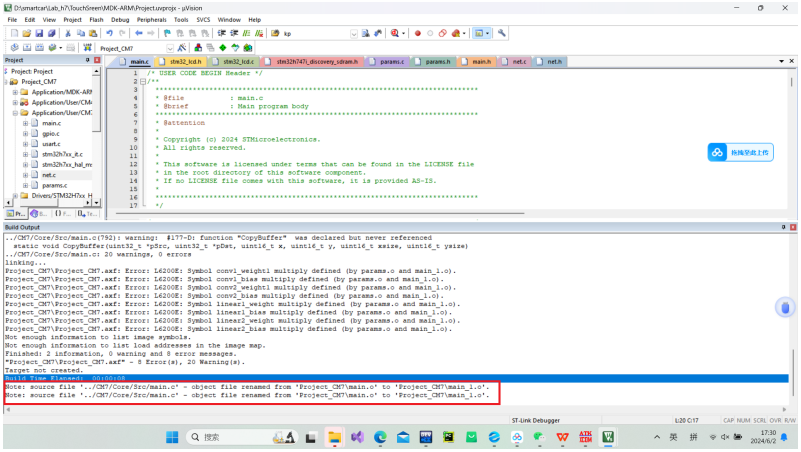


图 11: problem1

出现了 main 文件重名与内存不足的原因。经分析，我们发现我们的算法代码文件超出了 H747 所具有的空间，因此其无法支持我们自制识别的 C 语言代码，说明使用 CubeAI 非常必要。

5.2 cubeAI 相关文件路径错误

我们在 ioc 中配置了 cubeAI 相关的设置并生成代码。但是在 keil 编译中出现了错误路径的报错。随后发现是由于工程文件名与 ioc 的名称不同导致的错误，在修改名称后，问题解决。

5.3 数组元素序号错误

在我们把模型成功使用 cubeAI 部署到板子上之后发现，即使在本机上识别效果很好，但在板子上性能降低特别多，出现大量识别错误的情况。于是我们寻找问题，思考是否是模型转换成 onnx 时出现问题，或是模型的调用出现问题，所以我们更换了识别模型总共尝试了三个不同的模型，但依旧出现大量错误且没有规律可言。

随后我们检查了 keil 里的代码，发现在数组赋值部分，代码 “$indata[(TSState.TouchX-24)/10*28+(TSState.TouchX-8)/10]=(float)255$” 出现了错误。

首先，横纵坐标的表示都误用了 TouchX，应该横坐标对应 TouchX，纵坐标对应 TouchY。其次，将二维数组转换成一维数组时，28x28 中 n 行 m 列（从 0 开始计数）的元素对应在一维数组中的序号数应该为 n*28+m，所以应在 TouchY 对应的部分乘以 28。此外，根据 if 判断条件可知改触摸屏的 TouchX 范围为

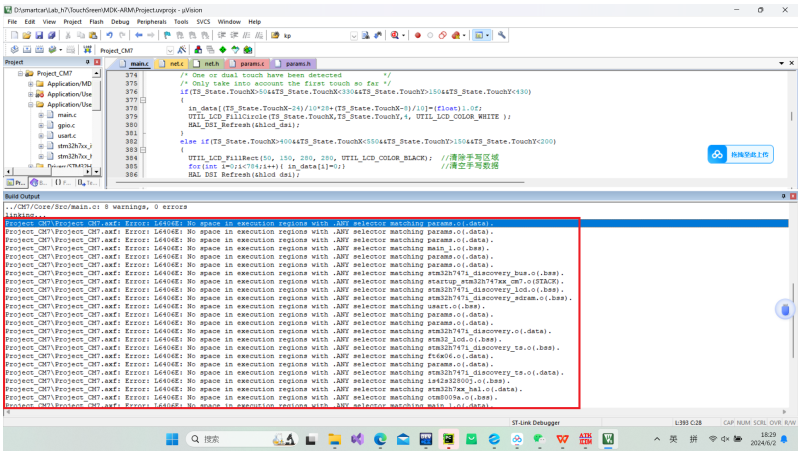
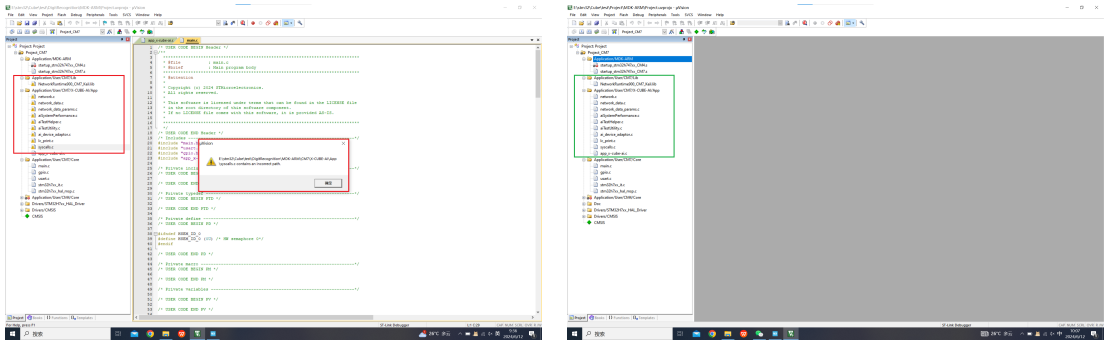


图 12: problem2



(a) (b)

图 13: problem3

50 到 330，TouchY 的范围为 150 到 430，因此在获取行列序号时应 TouchX-50 再除 10，TouchY-150 再除 10。

将该部分代码改成 “$in_data[(TS_State.TouchX-50)/10+((TS_State.TouchY-150)/10)*28] = (float)255;$”，能够在数组正确的对应位置进行赋值操作。同时，为了避免像素块过小导致识别不精准等问题，我们把检测到触碰的像素块及其周围的共四个像素块都赋值为 1。

```
if(TS_State.TouchX>50&&TS_State.TouchX<330&&TS_State.TouchY>150&&TS_State.TouchY<430)
{
  in_data[(TS_State.TouchX-24)/10*28+(TS_State.TouchX-8)/10]=(float)255;
  UTIL_LCD_FillCircle(TS_State.TouchX,TS_State.TouchY,4, UTIL_LCD_COLOR_WHITE );
  HAL_DSI_Refresh(&hlcd_dsi);
}
```

图 14: problem4

6 附录 源代码部分

6.1 pytorch 模型代码

```python
from imports import *

class net(torch.nn.Module):
    def __init__(self):
        super(net, self).__init__()
        self.conv1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 10, kernel_size=5),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2)
        )
        self.conv2 = torch.nn.Sequential(
            torch.nn.Conv2d(10, 20, kernel_size=5),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2)
        )
        self.fc = torch.nn.Linear(320, 10)
    def forward(self, x): #x是输入的图像
        batch_size = x.size(0)
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(batch_size, -1) #压平转1维
        x = self.fc(x) #10维矩阵, 对应0-9
        return x
```

6.2 pt 参数文件转 onnx 参数文件

```python
import torch

# 加载训练好的 PyTorch 模型
model_path = "model.pt"
model = torch.load(model_path)
model.eval()  # 设置模型为评估模式

# 将模型移动到 GPU上
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# 定义模型的输入
dummy_input = torch.randn(1, 1, 28, 28).to(device)

# 导出为 ONNX
onnx_path = "model.onnx"
torch.onnx.export(
    model,                  # 要转换的模型
    dummy_input,            # 模型输入示例
    onnx_path,              # 输出文件路径
    export_params=True,     # 保存模型参数
    opset_version=11,       # ONNX 版本
    do_constant_folding=True,  # 是否执行常量折叠优化
    input_names=['input'],      # 输入节点名称
    output_names=['output'],     # 输出节点名称
    dynamic_axes={'input': {0: 'batch_size'}, 'output': {0: 'batch_size'}}  # 动态轴设置
)

print(f"模型已成功导出为 {onnx_path}")
```

6.3 Core7–main 函数核心部分

```
  /* USER CODE BEGIN WHILE */
  while (1)
  {
//      /* Check in polling mode in touch screen the touch status and coordinates */
//      /* of touches if touch occurred                                           */
      ts_status = BSP_TS_GetState(0, &TS_State);
      if(TS_State.TouchDetected)
      {
        /* One or dual touch have been detected          */
        /* Only take into account the first touch so far */
        if(TS_State.TouchX>50&&TS_State.TouchX<330&&TS_State.TouchY>150&&TS_State.TouchY
    <430)
        {
          in_data[(TS_State.TouchX-50)/10+((TS_State.TouchY-150)/10)*28]=(float)255;
          in_data[((TS_State.TouchX-50)/10+1)+((TS_State.TouchY-150)/10)*28]=(float)255;
          in_data[(TS_State.TouchX-50)/10+((TS_State.TouchY-150)/10+1)*28]=(float)255;
          in_data[((TS_State.TouchX-50)/10+1)+((TS_State.TouchY-150)/10+1)*28]=(float)255;

          UTIL_LCD_FillCircle(TS_State.TouchX,TS_State.TouchY,4, UTIL_LCD_COLOR_WHITE );
          HAL_DSI_Refresh(&hlcd_dsi);
        }
        else if(TS_State.TouchX>400&&TS_State.TouchX<550&&TS_State.TouchY>150&&TS_State.
    TouchY<200)
        {
          UTIL_LCD_FillRect(50, 150, 280, 280, UTIL_LCD_COLOR_BLACK);
          for(int i=0;i<784;i++){ in_data[i]=0;}
          HAL_DSI_Refresh(&hlcd_dsi);
          UTIL_LCD_FillRect(600, 350, 100, 100, UTIL_LCD_COLOR_WHITE);
        }
        else if(TS_State.TouchX>400&&TS_State.TouchX<550&&TS_State.TouchY>250&&TS_State.
    TouchY<300)
        {
          //???f ?D'?se??•Í??'y  3 ?
          for(int i=0;i<784;i++){
            input_data[i]=in_data[i];
          }
          AI_Run();
          /*char s[]="0";
          int i,j,a=0,b=0;
          int k[28]={0};
          for(i=0;i<28;i++){
            for(j=0;j<28;j++){
              if(in_data[i*28+j]==1)
                k[i]++;
            }
            if(k[i]>1)
              a++;
          }*/
          float t;
          int num;
          num=0;
          t=output_data[0];
          for(int i;i<10;i++){
            if(output_data[i]>t){
              t=output_data[i];
              num=i;
```

```
            }
          }
          HAL_DSI_Refresh(&hlcd_dsi);
          if(num==0)
            UTIL_LCD_DisplayStringAt(600,350,(uint8_t*) "0", LEFT_MODE);
          else if(num==1)
            UTIL_LCD_DisplayStringAt(600,350,(uint8_t*) "1", LEFT_MODE);
          else if(num==2)
            UTIL_LCD_DisplayStringAt(600,350,(uint8_t*) "2", LEFT_MODE);
          else if(num==3)
            UTIL_LCD_DisplayStringAt(600,350,(uint8_t*) "3", LEFT_MODE);
          else if(num==4)
            UTIL_LCD_DisplayStringAt(600,350,(uint8_t*) "4", LEFT_MODE);
          else if(num==5)
            UTIL_LCD_DisplayStringAt(600,350,(uint8_t*) "5", LEFT_MODE);
          else if(num==6)
            UTIL_LCD_DisplayStringAt(600,350,(uint8_t*) "6", LEFT_MODE);
          else if(num==7)
            UTIL_LCD_DisplayStringAt(600,350,(uint8_t*) "7", LEFT_MODE);
          else if(num==8)
            UTIL_LCD_DisplayStringAt(600,350,(uint8_t*) "8", LEFT_MODE);
          else
            UTIL_LCD_DisplayStringAt(600,350,(uint8_t*) "9", LEFT_MODE);

          HAL_Delay(500);  //ś ? ař'??ř'??? '???? '? 'DD?ď2a??
          HAL_DSI_Refresh(&hlcd_dsi);
        }
      }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```