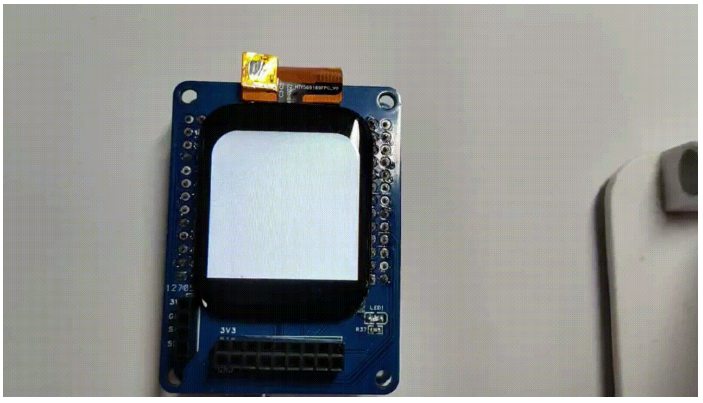
手写数字识别 MNIST Demo

该例程使用触摸屏p169h002-ctp，在LCD上进行手写数字，再进行手写数字的识别。动态演示图如下所示：



MNIST是一个非常经典的机器学习的入门案例，本demo将会使用keras进行训练，导出模型部署到FryPie炸鸡派上。

> MNIST数据库是手写数字的大型数据库，通常用于训练各种图像处理系统。

一、文件夹组成

```
├─python_codes
│   │  mnist.npz
│   │  mnist.h5
│   │  train.py
│
└─stm32_codes
    ├─MNIST
```

文件夹大致如上，`python_codes`存放的是：train.py用于训练网络，`mnist.npz`为网上找的下载的MNIST数据集，用于网络的训练和测试，最后训练好的模型保存为`mnist.h5`。

二、触摸屏手写显示在LCD

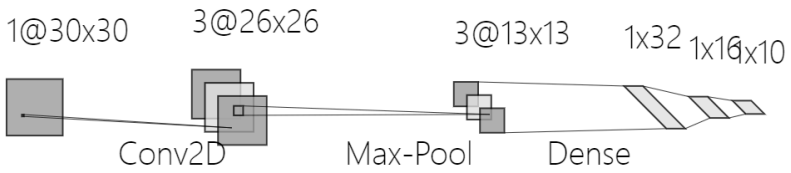
屏幕的大小是240×280，使用上240×240部分作为绘制板，用于手写显示，下面240×40的部分用于显示识别的结果。读取点与绘制如下代码所示。由于CST816读取频率问题，绘制的触摸点与点的距离会有一定距离，同时1×1的点非常小，因此，需要对点进行膨胀，都变为8×8的黑点绘制到LCD上，其中`SCALL_RATE`为8。同时，这里也要完成`originalImage`数组的填充，作为原始的图片数据。

当然，还有一个问题，如果手写滑动速度太快，就算扩充成了8×8，点与点的距离远了看起来还是很明显，所以推荐再做处理，将没相连的黑点连接在一起。

```
if(CST816_Get_FingerNum()!=0x00 && CST816_Get_FingerNum()!=0xFF)
{
      if(!reg_state)
      {
        CST816_Get_XY_AXIS();
        if(CST816_Instance.X_Pos > SCALL_RATE/2 && CST816_Instance.Y_Pos >
SCALL_RATE/2)
        {
          for(uint8_t i=0; i<SCALL_RATE; i++)
          {
            for(uint8_t j = 0; j < SCALL_RATE; j++)
            {
              originalImage[(CST816_Instance.X_Pos - SCALL_RATE/2 + j) +
                            (CST816_Instance.Y_Pos - SCALL_RATE/2 + i)*240] = 0xFF;
            }
          }
          LCD_Fill(CST816_Instance.X_Pos - SCALL_RATE/2, CST816_Instance.Y_Pos -
SCALL_RATE/2,
                   CST816_Instance.X_Pos + SCALL_RATE/2, CST816_Instance.Y_Pos +
SCALL_RATE/2, BLACK);
        }
      }
}
```

三、模型训练

搭建的模型如下所示，使用的手写数字MNIST数据集在网上很多，可以在kaggle上直接找数据集。训练完生成.h5模型最后需要部署到STM32上。




```
#------------------------------【搭模型】--------------------------------
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters=3, kernel_size=(5, 5), padding='valid',
activation=tf.nn.relu, input_shape=(30, 30, 1)),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2), padding='same'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=32, activation=tf.nn.relu),
    tf.keras.layers.Dense(units=16, activation=tf.nn.relu),
    tf.keras.layers.Dense(units=10, activation=tf.nn.softmax)
])

model.summary()
```

四、数据处理，模型部署

由于数据量的问题，需要吧240×240的数据压缩成30×30的数据，不然模型的参数量会非常的庞大，STM32上根本跑不了。下采样如下代码所示，这里是一个平均值池化，将240×240的图像数据压缩为30×30，原来的`originalImage[]`为`uint8_t`类型，压缩的`compressedImage[]`为float型。最后再将`compressedImage[]`输入到模型的API接口，得到输出找到最大值对应的index即为识别到的数字。

```
void downsample(uint8_t * originalImage, float * compressedImage)
{
    for (int i = 0; i < COMPRESSED_SIZE; i++) {
        // 计算原始图片区域的左上角坐标
        int start_x = (i % 30) * (8); // 每行240个像素压缩成30个像素
        int start_y = (i / 30) * (8); // 每列240个像素压缩成30个像素

        // 计算原始图片区域的右下角坐标
        int end_x = start_x + 8; // 30 / 240 = 0.125, 8 = 240 * 0.125
        int end_y = start_y + 8;

        // 计算原始图片区域内像素的平均值
        int sum = 0;
        for (int x = start_x; x < end_x; x++) {
            for (int y = start_y; y < end_y; y++) {
                sum += originalImage[x + y * 240];
            }
        }
        compressedImage[i] = sum / 64; // 8*8=64
        //norm
```

```
            compressedImage[i] /= 255;
        }
    }
```