

Python实训开发——手写数字识别项目汇报

1.项目简介

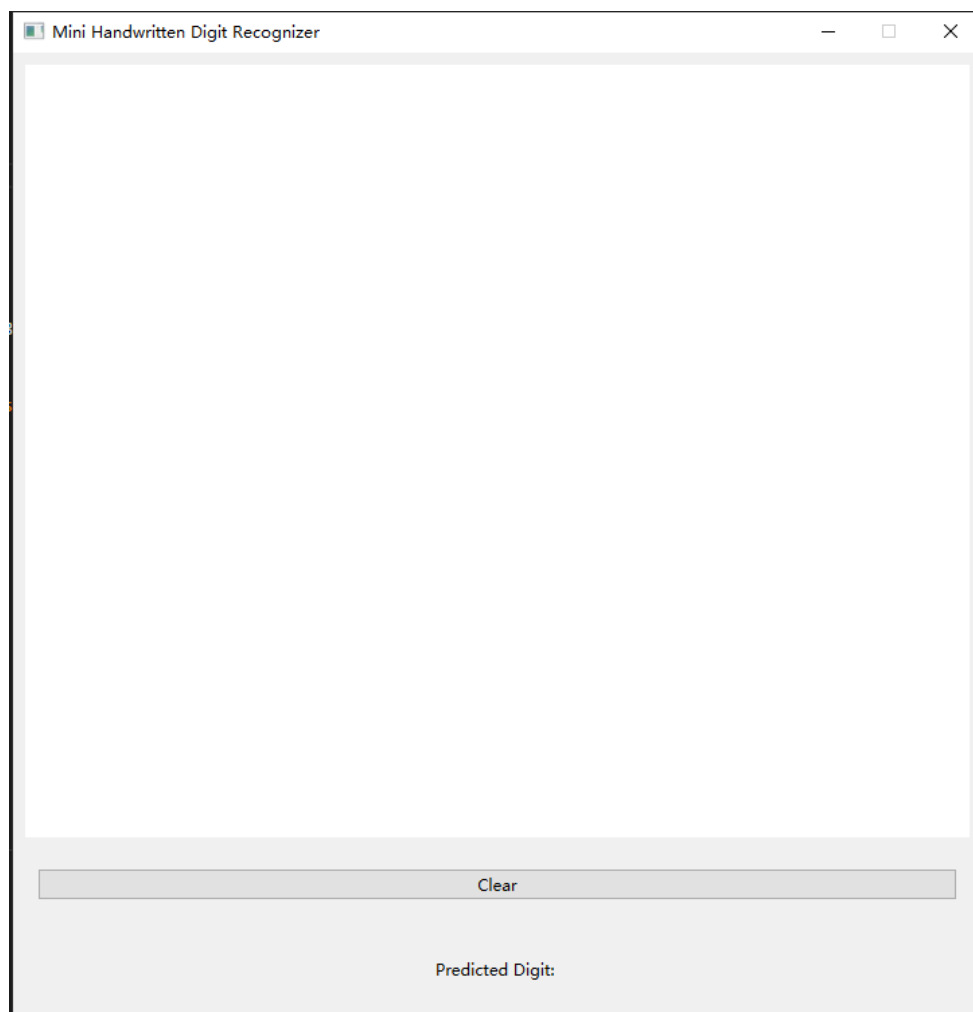
项目代码: [Crist1128/Handwritten-numeral-recognition: A simple handwritten digit recognition tool written on tensorflow \(github.com\)](https://github.com/Crist1128/Handwritten-numeral-recognition)

本项目是一个手写数字识别应用，基于深度学习技术实现。它提供了一个用户友好的图形界面，允许用户在画布上手写数字，然后通过预训练的卷积神经网络模型对绘制的数字进行识别。这个项目具有以下主要功能：

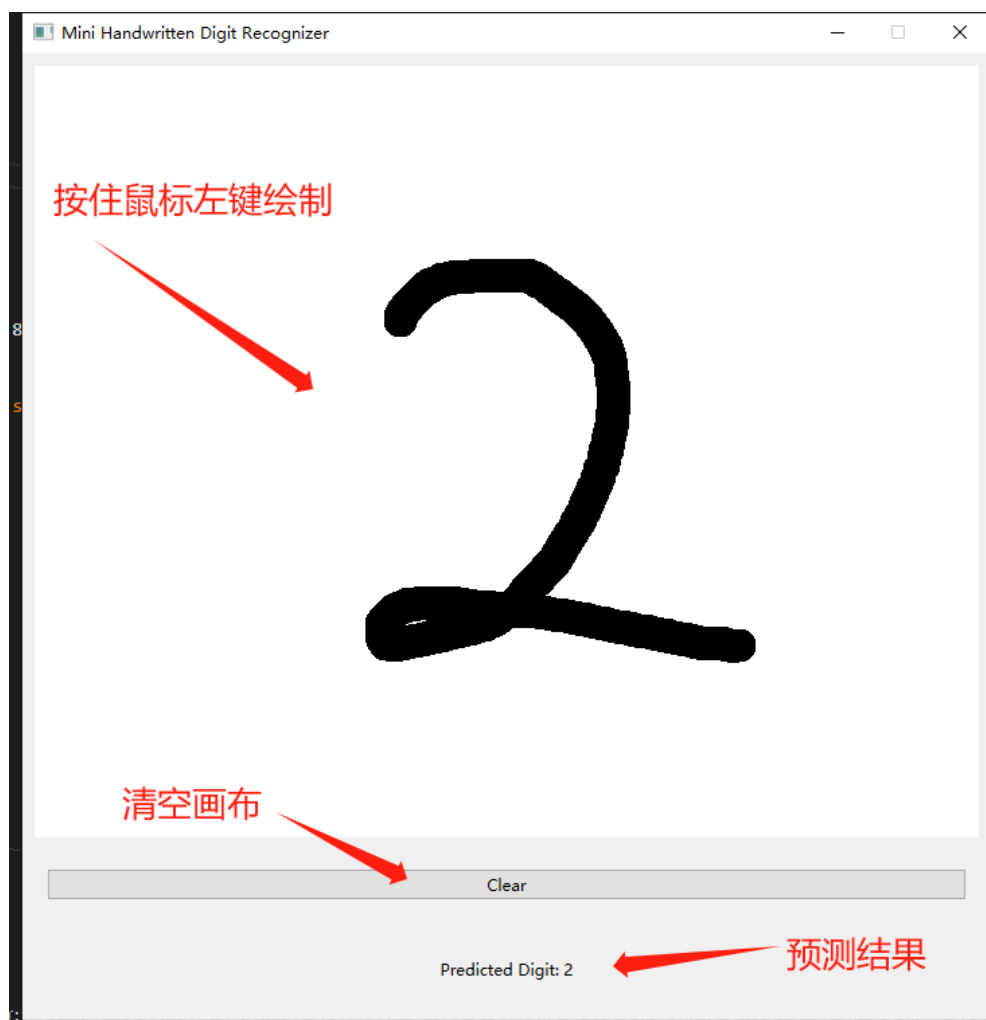
1. 用户可以在画布上绘制手写数字。
2. 项目提供了一个清除按钮，用户可以清空画布以重新绘制数字。
3. 绘制的数字将通过一个预训练的深度学习模型进行识别。
4. 识别结果将显示在界面上，帮助用户了解他们绘制的数字。

2.项目成果

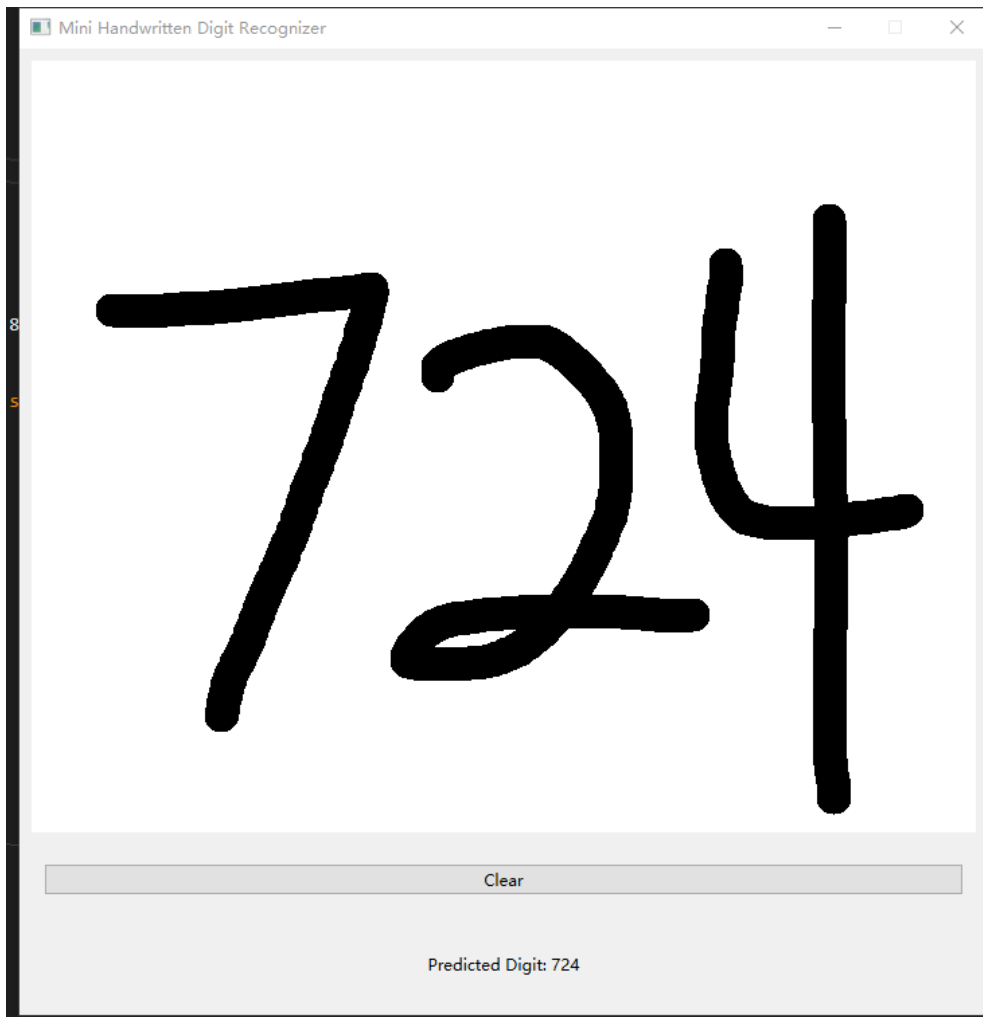
- 运行主页面：



- 单数字预测样例



- 支持多数字识别



3.系统开发

3.1 开发环境和工具

1. **编程语言**：项目主要使用了Python编程语言
2. **图形用户界面库**：项目使用了PySide6库来创建用户友好的图形用户界面。
3. **深度学习框架**：深度学习模型的开发和训练使用了TensorFlow和Keras。TensorFlow是一个开源的深度学习框架，而Keras是一个高级神经网络API，它与TensorFlow集成在一起，用于创建和训练深度学习模型。
4. **图像处理库**：Python的PIL库（Python Imaging Library），进行图像处理操作，包括图像的转换、调整大小和保存。
5. **数据集**：在模型的训练阶段，使用了MNIST手写数字数据集，这是一个包含大量手写数字图像和相应标签的经典数据集。
6. **集成开发环境（IDE）**：Visual Studio Code
7. **操作系统**：Windows10
8. **版本控制**：如果你使用了版本控制工具（如Git）来管理项目的代码，也可以在这里提及。

3.2 核心模块的实现思路

3.2.1 CNN细节

采用了卷积神经网络进行模型的训练，准确率为99.3%（可能过拟合）

`build_model()` 函数：

- 创建一个名为 `model` 的Sequential模型，这个模型是一个层的线性堆叠。
- `Conv2D` 层：添加第一个卷积层，使用32个大小为(3, 3)的过滤器，激活函数为ReLU（Rectified Linear Unit），并指定输入数据的形状为28x28像素的RGB图像（3通道）。
- `MaxPooling2D` 层：添加最大池化层，用于减小图像的空间尺寸，提取特征。
- 再次添加一个 `Conv2D` 层，这次使用64个过滤器。
- 再次添加一个最大池化层。
- `Flatten` 层：展平层，用于将卷积层的输出展平为一维向量，以供全连接层使用。
- `Dense` 层：添加一个全连接层，包含64个神经元，激活函数为ReLU。
- `Dropout` 层：添加Dropout层，以减少过拟合风险。Dropout是一种正则化技术，随机丢弃一部分神经元的输出。
- 再次添加一个 `Dense` 层，输出层，包含10个神经元，激活函数为Softmax，用于多类别分类。
- 最后，使用 `model.compile()` 编译模型，指定优化器（Adam）、损失函数（交叉熵损失函数）和评估指标（准确度）。
- 返回构建好的深度学习模型。

```
def build_model():
    model = Sequential()

    #两个卷积层，分别有32、64个过滤器
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 3)))
    model.add(MaxPooling2D((2, 2)))#最大池化层
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))

    model.add(Flatten())#展平层
    model.add(Dense(64, activation='relu'))#全连接层
    model.add(Dropout(0.5))#减少过拟合风险
    model.add(Dense(10, activation='softmax')) # 全连接层，10个数字类别

    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=
['accuracy'])

    return model
```

3.2.2 延迟识别

延迟保存图片的实现是通过定时器（QTimer）实现的。具体实现步骤如下：

1. 当用户开始绘制时，鼠标移动事件（`mouseMoveEvent`）被触发。
2. 在 `mouseMoveEvent` 方法中，如果用户按下了鼠标左键，那么启动了一个定时器 `self.timer`，该定时器被设置为一次性触发，并在1秒后触发 `self.save_drawing` 方法。

这个定时器的目的是延迟保存图片，以确保用户完成绘制后才保存绘制的结果。用户在绘制时，可能需要一些时间来完成他们的手写数字，所以等待一段时间后再保存图片是为了确保用户完成了绘制。一旦用户完成绘制，定时器触发，执行 `self.save_drawing` 方法，将绘制的图像进行处理和保存。这种方式确保了图片的延迟保存。

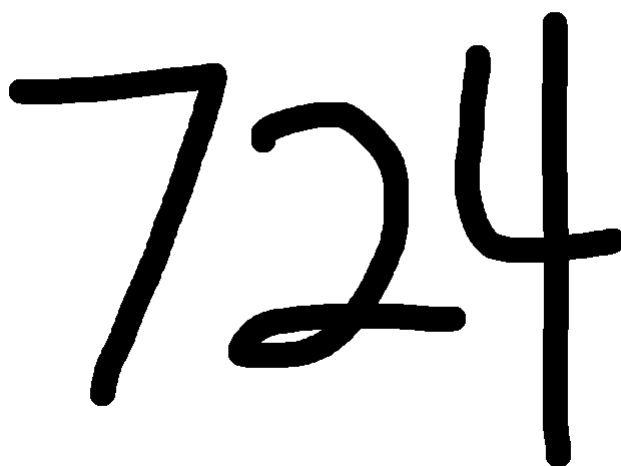
3.2.3 多数字识别

多数字识别主要是通过类似“抠图”的方法实现的，关键代码是：

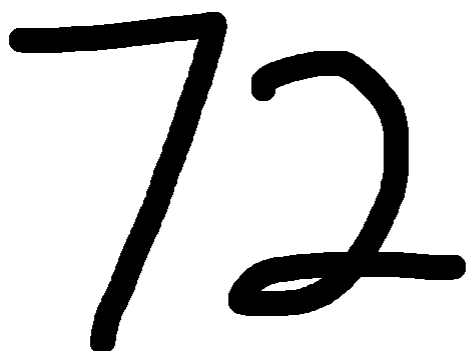
```
for i in range(image_array.shape[0]):
    for j in range(image_array.shape[1]):
        num_now = image_array[i, j]
        num_last = self.last_drawing[i, j]
        if num_now <= 127 and num_last <= 127:
            image_array[i, j] = 255

self.last_drawing = np.array(image)
```

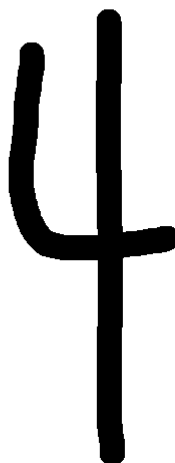
对于样例中的“724”，在识别时，我们采用当前绘制图片：

A large, bold, black handwritten number '724' on a white background. The '7' is a simple vertical stroke with a short horizontal top bar. The '2' is a cursive shape with a loop at the bottom. The '4' is a simple vertical stroke with a horizontal crossbar.

减去（采用了类似相减的思路，具体可以看代码内容）上一次绘制图片：

A large, bold, black handwritten number '72' on a white background. The '7' is a simple vertical stroke with a short horizontal top bar. The '2' is a cursive shape with a loop at the bottom.

从而得出本次绘制的数字：



对该图片进行转换后输入模型进行预测

本质上还是单数字识别，但是可以模拟多数字的情形

3.3 项目缺陷

在项目中有以下一些功能未能实现：

- 对“抠图”后的数字进行居中后再进行识别
 - 当手写数字处于画布边缘时，识别效果较差。这使得我们在进行多数字识别时会导致靠近边缘的数字识别错误率较高
- 实现真正意义上的多数字识别

4.源码解析

程序主要包含以下四个python文件：

- model.py
- train.py
- canvas.py
- main.py

1. model.py:

- 定义了深度学习模型的结构，用于手写数字识别。模型使用了Keras库，包括卷积神经网络（Convolutional Neural Network, CNN）的定义。
- 主要作用：构建神经网络模型，定义了模型的架构，包括卷积层、池化层、全连接层和Dropout层，并编译了模型以准备训练。

2. train.py:

- 包含了用于训练深度学习模型的代码。
- 主要作用：加载和预处理手写数字数据集，创建模型，进行模型的训练，然后将训练好的模型保存到文件夹下。

3. canvas.py:

- 定义了一个用户界面部件，即绘制数字的画布，允许用户在画布上绘制手写数字。

- 主要作用：管理用户的鼠标操作，包括绘制、清除和保存绘制的图像。还处理了图像的预处理步骤，以便进行数字识别。

4. main.py:

- 这是主应用程序文件，创建了一个图形用户界面，整合了画布、清除按钮和显示预测结果的标签。
- 主要作用：初始化主窗口，与画布部件和清除按钮部件进行交互，加载预训练的深度学习模型，预测用户绘制的手写数字，并在界面上显示识别结果。

4.1 model.py源码解析

导入的包:

- `import tensorflow as tf`: 引入TensorFlow库，用于构建深度学习模型和进行训练。
- `from tensorflow import keras as kr`: 引入Keras库的一部分，用于构建深度学习模型。
- `from keras.models import Sequential`: 从Keras中引入Sequential模型，这是一种用于构建深度学习模型的顺序模型。
- `from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout`: 从Keras中引入用于构建卷积神经网络的不同层类型，包括卷积层、池化层、全连接层和Dropout层。

`build_model()` 函数:

- 这个函数用于构建深度学习模型，主要构建一个卷积神经网络（Convolutional Neural Network, CNN）。
- 创建一个名为 `model` 的Sequential模型，这个模型是一个层的线性堆叠。
- `Conv2D` 层: 添加第一个卷积层，使用32个大小为(3, 3)的过滤器，激活函数为ReLU（Rectified Linear Unit），并指定输入数据的形状为28x28像素的RGB图像（3通道）。
- `MaxPooling2D` 层: 添加最大池化层，用于减小图像的空间尺寸，提取特征。
- 再次添加一个 `Conv2D` 层，这次使用64个过滤器。
- 再次添加一个最大池化层。
- `Flatten` 层: 展平层，用于将卷积层的输出展平为一维向量，以供全连接层使用。
- `Dense` 层: 添加一个全连接层，包含64个神经元，激活函数为ReLU。
- `Dropout` 层: 添加Dropout层，以减少过拟合风险。Dropout是一种正则化技术，随机丢弃一部分神经元的输出。
- 再次添加一个 `Dense` 层，输出层，包含10个神经元，激活函数为Softmax，用于多类别分类。
- 最后，使用 `model.compile()` 编译模型，指定优化器（Adam）、损失函数（交叉熵损失函数）和评估指标（准确度）。
- 返回构建好的深度学习模型。

```
# model.py
import tensorflow as tf
from tensorflow import keras as kr
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

def build_model():
    model = Sequential()
```

```

#两个卷积层，分别有32、64个过滤器
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 3)))
model.add(MaxPooling2D((2, 2)))#最大池化层
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())#展平层
model.add(Dense(64, activation='relu'))#全连接层
model.add(Dropout(0.5))#减少过拟合风险
model.add(Dense(10, activation='softmax')) # 全连接层，10个数字类别

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=
['accuracy'])

return model

```

这个模型是一个常见的手写数字识别模型，通过卷积神经网络对输入的手写数字图像进行特征提取和分类。它是一个顺序的模型，按顺序堆叠了各种神经网络层，最后通过Softmax激活函数输出对10个数字类别的概率分布。

4.2 train.py源码解析

导入的包：

- `from data_processing import load_and_preprocess_data` 和 `from model import build_model`：

这两行代码导入了之前在项目中定义的 `data_processing` 模块和 `model` 模块，用于数据处理和模型构建的相关功能。

其余代码：

1. `data_dir` 变量：
 - 这个变量包含了手写数字数据集的路径，它指定了数据集所在的文件夹位置。
2. `load_and_preprocess_data(data_dir)`：
 - 调用 `data_processing` 模块中的 `load_and_preprocess_data` 函数，用于加载和预处理手写数字数据集。这个函数的作用是从指定路径加载数据集，进行必要的的数据预处理操作，并返回准备好的训练数据。
3. `build_model()`：
 - 调用 `model` 模块中的 `build_model` 函数，用于构建深度学习模型。这个函数已经在之前的解析中详细介绍过，它返回一个已经定义好架构的模型。
4. `model.fit(train_data, epochs=6)`：
 - 使用训练数据 `train_data` 对构建好的深度学习模型进行训练。`model.fit` 方法将模型与训练数据进行拟合，训练模型的权重和参数。
 - `epochs=6` 指定了进行训练的周期数，这里是6个周期。
5. `model.save('mnist_model_6.h5')`：
 - 训练完成后，这行代码将训练好的模型保存到磁盘上的文件 `'mnist_model_6.h5'` 中。这个文件将包含模型的权重、架构和配置，以便将来在应用中加载和使用模型。

```
# train.py
```



```

from data_processing import load_and_preprocess_data
from model import build_model

# 数据路径
data_dir = "E:/pythontraining/MNIST/train"

# 加载和预处理数据
train_data = load_and_preprocess_data(data_dir)

# 构建模型
model = build_model()

# 训练模型
model.fit(train_data, epochs=6)

# 保存模型
model.save('mnist_model_6.h5')

```

`train.py` 的作用是加载和预处理手写数字数据集，构建深度学习模型，进行模型训练，并将训练好的模型保存到磁盘，为手写数字识别应用提供了训练好的模型。

4.3 canvas.py源码解析

`canvas.py` 文件主要负责创建一个可交互的绘图画布，允许用户在上面手写数字，并实现了与用户交互和数字图像处理相关的功能。以下是对这个文件的主要功能和每个方法的解析：

主要功能：

`canvas.py` 主要提供以下功能：

1. 创建一个绘图画布，允许用户在上面手写数字。
2. 监听用户鼠标事件，实现绘制数字、清除画布、保存绘制的图像和与主窗口的通信。
 - 实现了延迟识别（在用户鼠标停止绘制时，将自动进行识别）
 - 实现了多数字识别
3. 对用户绘制的数字图像进行一系列的预处理，以便后续的数字识别。

方法解析：

1. `__init__(self, main_window)` 方法：
 - 初始化画布部件，接受一个指向主窗口的引用，以便与主窗口进行通信。
 - 创建一个空白的画布 `QPixmap`，并设置画布大小与部件大小相同。
 - 初始化定时器，用于触发保存绘制的图像。
 - 初始化变量 `last_x` 和 `last_y` 用于跟踪鼠标绘制路径。
 - 初始化 `last_drawing` 和 `cnt` 变量，用于图像处理。
2. `paintEvent(self, event)` 方法：
 - 当部件需要重绘时，此方法负责在画布上绘制内容。使用 `QPainter` 绘制当前画布上的内容。
3. `resizeEvent(self, event)` 方法：

- 当部件大小变化时，此方法负责调整画布的大小以匹配新的部件大小，并重新初始化画布和相关变量。

4. `mouseMoveEvent(self, event)` 方法：

- 监听鼠标移动事件，允许用户在画布上绘制数字。
- 当鼠标左键按下时，记录绘制路径，设置笔刷样式，绘制线条，触发更新部件以实时显示绘制内容，并启动定时器以便保存绘制结果。

5. `mouseReleaseEvent(self, event)` 方法：

- 监听鼠标释放事件，停止定时器，准备保存绘制的图像，重置绘制状态。

6. `clear(self)` 方法：

- 清空画布，填充为白色。
- 重置绘制状态，清空预测结果数组，触发更新部件以清空画布内容。

7. `save_drawing(self)` 方法：

- 保存用户绘制的图像，进行一系列的图像处理操作，包括转换为灰度图像、二值化、调整大小和保存图像文件。
- 调用主窗口的 `predict_digit` 方法，以用于数字识别。

`canvas.py` 提供了用户友好的绘图界面，允许用户在画布上绘制数字，并实时更新绘制内容。它还处理了绘制结果的保存和预处理，以便进行数字识别。

4.4 main.py源码解析

主函数 (main.py):

- 这是项目的主要应用程序文件，用于创建一个图形用户界面 (GUI) 来进行手写数字识别。
- 通过导入不同的包和库，创建了一个交互式界面，允许用户手写数字并获取识别结果。

类定义 - MainWindow:

- `MainWindow` 类是项目的主窗口，继承自 `QMainWindow`，用于创建应用程序的主界面。
- 在初始化函数 `__init__` 中，进行了以下关键操作：
 - 设置窗口的标题和大小。
 - 创建一个空的列表 `predicted_digits`，用于存储多次识别的结果。
 - 创建了一个用于绘制的画布部件 `self.drawing_canvas`，并将其添加到主窗口。
 - 创建了一个清除按钮 `clear_button`，用于清空绘制的内容，然后将其添加到主窗口。
 - 设置主布局，包括画布和清除按钮。
 - 连接清除按钮的点击事件到处理函数 `clear_drawing`。
 - 加载预训练的深度学习模型 (`'mnist_model_6.h5'`)，该模型用于手写数字的识别。
 - 创建一个标签 `self.result_label`，用于显示预测的数字，将其添加到主布局中。
- `clear_drawing` 方法用于处理清除按钮的点击事件，它清空了绘制的内容并重置 `predicted_digits` 列表。
- `predict_digit` 方法用于预测手写数字并更新结果标签。它接受一个图像，将其传递给预训练的深度学习模型，获取预测结果，并将结果显示在 `self.result_label` 中。

主函数 (__main__):

- 在 `__main__` 部分，首先创建一个 `QApplication` 应用程序实例。

- 然后创建主窗口 `window`，并显示出来。
- 最后，通过 `app.exec()` 启动应用程序的事件循环，以等待用户与界面交互。

5.总结

项目总结：

本项目是一个手写数字识别应用，基于深度学习技术，提供了用户友好的图形用户界面，允许用户在画布上手写数字，并通过预训练的卷积神经网络模型进行数字识别。项目通过以下主要功能实现：

1. **用户友好的界面**：项目提供了一个清晰、易于使用的图形用户界面，使用户可以轻松地绘制手写数字。
2. **多数字识别**：尽管本质上是单数字识别，项目实现了多数字的模拟识别，通过"抠图"的方式，允许用户在同一画布上绘制多个数字，并分别进行识别。
3. **深度学习模型**：构建了一个卷积神经网络模型，通过训练和预测，实现了高准确度的手写数字识别。

当然，项目还有改进的空间：

1. **多数字识别优化**：项目目前采用的多数字识别方式是一种模拟方法，可能会在数字之间产生干扰，建议改进以提高多数字识别的准确性。
2. **用户界面改进**：进一步改善用户界面设计，提高用户体验，可以考虑添加更多功能和交互性，如保存和加载手写数字、颜色选择等。
3. **异常处理**：添加更多的异常处理机制，以应对用户的非标准输入，提高系统的稳定性。
4. **可扩展性**：考虑将项目设计成可扩展的应用，允许将来添加更多功能或集成其他深度学习模型。

学习收获和未来展望：

通过这个项目，我学到了深度学习模型构建、图像处理、图形用户界面设计等关键技能。这些技能对于未来的数据科学、人工智能和应用开发都非常有用。未来我可以进一步扩展这个项目，将其用于更广泛的应用领域，如手写文字识别、数字数据处理等。还可以继续改进用户界面、优化模型性能，并提供更多高级功能。

总的来说，这个项目是一个对于python和深度学习来说非常有价值的实践项目，提供了一个实际的应用场景，以应用深度学习技术，在实践中学习往往是学习编程的最好方式。对这个项目的学习充实了我的项目经验，同时也对我的未来发展和项目有着积极的影响。