实验五：软件系统实现与测试

实验编号：**SE-005** 填写时间：**2023** 年 **6** 月 **2** 日

课程名称	软件工程实践				
姓名	张 子 扬	学号		班级	

实验目的和要求：

1.图片的管理功能

图片上传、删除和标定；

2.用户管理功能

用户登录，建立，查询，修改和删除；

3.图片分类功能实现

用户选择一副图片能识别图片数字内容；

4.模型训练参数设定

系统给出模型训练的批次、轮次、学习率等参数设定界面。

实验内容：

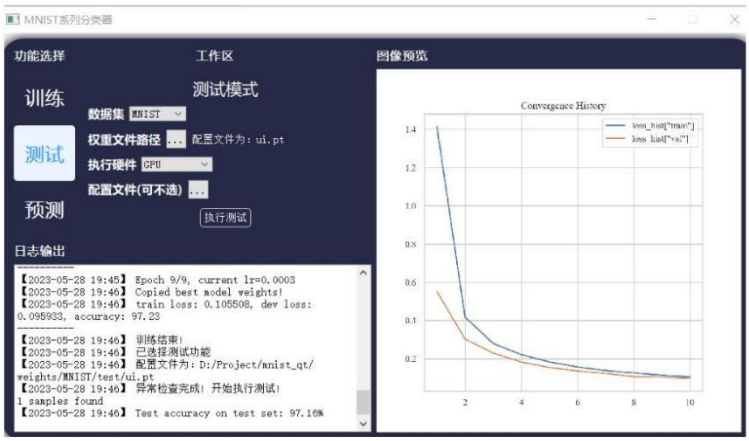
摘要：

本软件 GUI 部分由 PYQT5 实现，界面整体呈深蓝色，当用户鼠标悬停在任意按钮上时，按钮会变色。本软件下方设置了日志输出窗口，方便用户实时查看自己的操作输出。在界面的右侧设置了可视化窗口，方便用户查看训练过程中参数的可视化和预测的图像。

本软件除了实现了在 MNIST 数据集上的图像分类外，还实现了在 **FashionMNIST** 数据集上的分类。

本软件除了可以在可视化界面进行操作外，也可以一键导入自己的配置单，省去重复化操作。

本软件兼容性强，不仅可以在 **GPU** 和 **CPU** 环境上运行，经测试，还可以在 Linux 和 Windos 系统上进行运行。



除此之外，软件还有一个亮点，那就是对异常情况的处理。

异常情况主要分为两种，第一种就是系统资源不足，此时会启动多线程来用与系统争用资源，在测试中，即使在极端状况下，本软件几乎不会进入无响应状态！

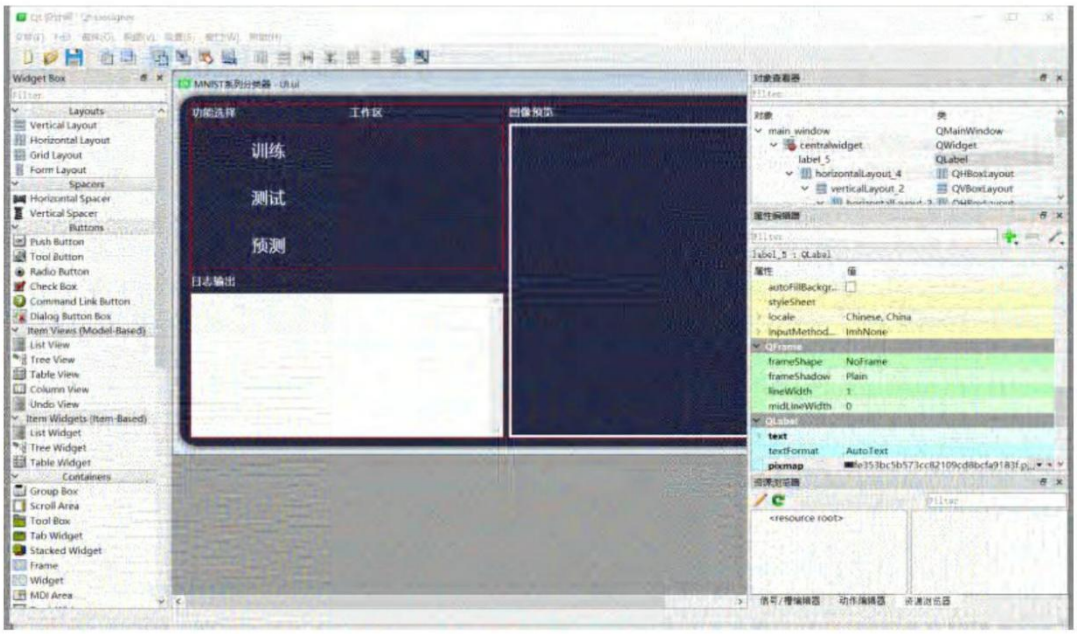
第二种情况是当软件进行训练或者测试的时候所有的按钮都会变为灰色不可选的状态。这样的设计有利于防止用户在这段时间进行误操作导致软件出现错误。

正文：

实验主要分为两个部分，分别是界面设计和深度学习工程。

首先介绍界面设计。

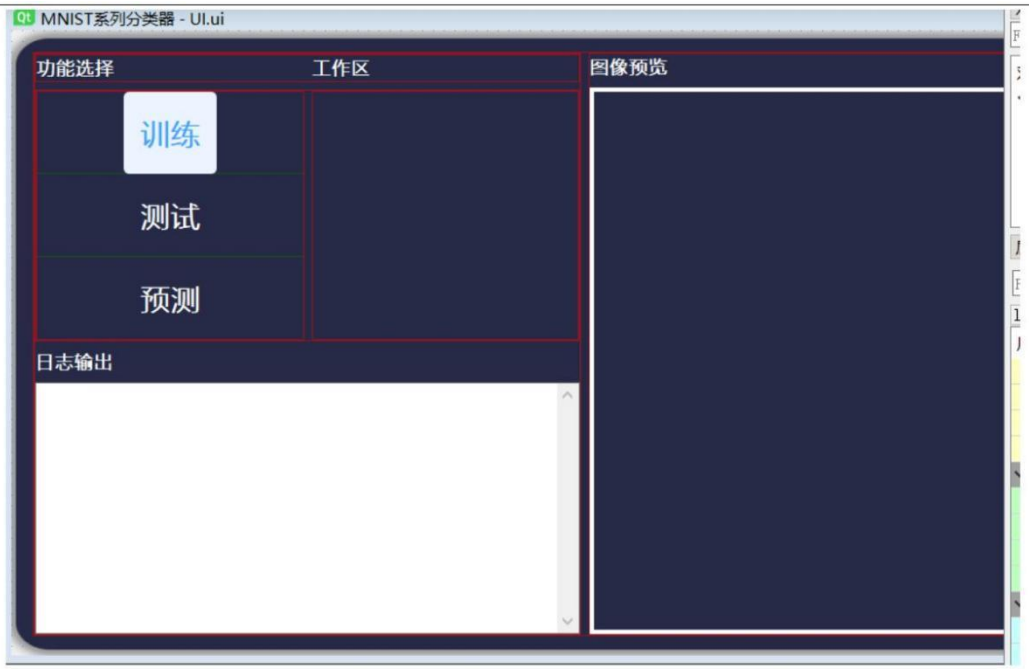
界面使用由 PYQT5 库提供的 Qt designer 软件设计，该软件提供了方便的图形化界面，可以通过拖拽的方式设计软件的界面。



在界面风格的选取上，我以深蓝色为主题来凸显软件的科技感和先进性。

在界面设计上，我在左侧设计了本软件的三个功能按钮，通过三个按钮可以在本软件提供的三个主要功能模块间进行切换。这三个功能分别是训练、测试、预测。

为了提高与用户的交互性，当用户的鼠标悬停在按钮上方时，按钮会变色同时光标将箭头变为手指。



这样，用户就可以清晰的看清自己光标所处的位置，可以避免用户的误操作，也可以满足用户对实时交互的需求。

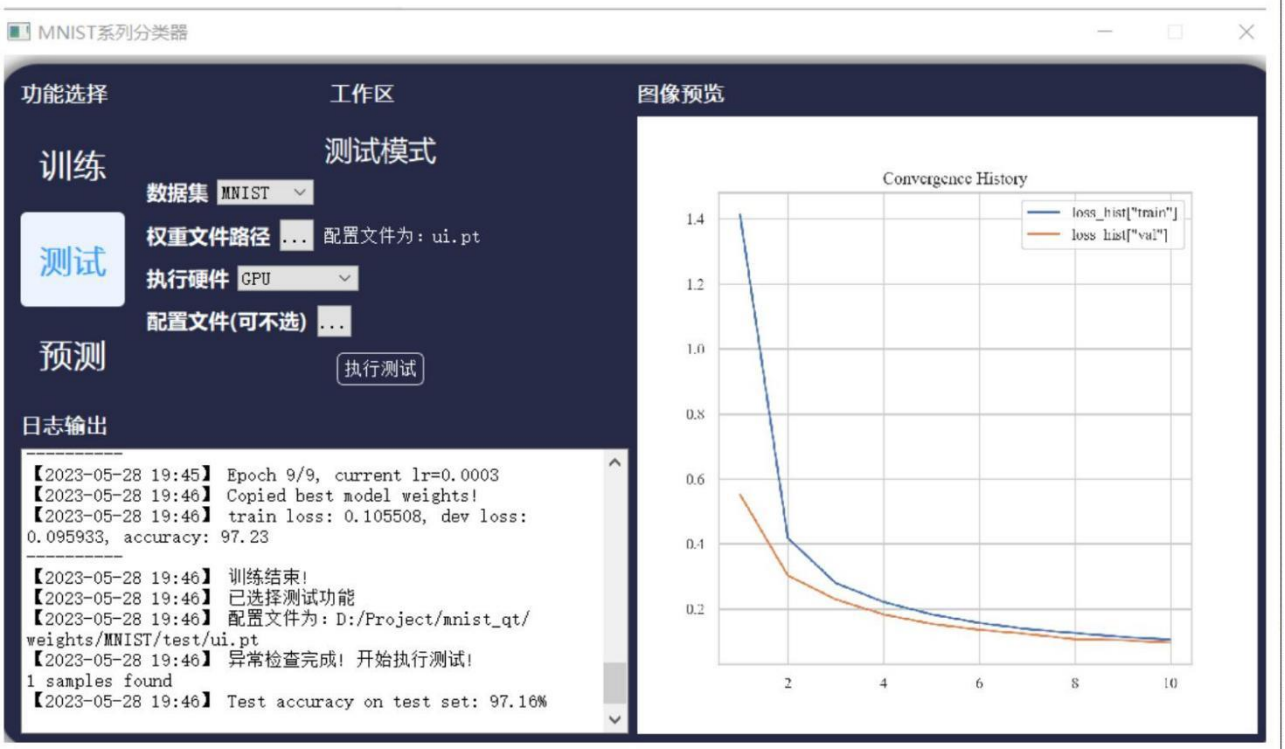
在软件的中央部分，我设计了工作区。当用户首次进入软件时，工作区会显示出软件名和软件的作者。

当用户选择三个主要的功能之一时，工作区会出现下拉菜单、文件选择、按钮等组件。

当选取训练功能时，工作区会出现如下四个选项，分别是“数据集”、“权重文件输出路径”、“执行硬件”、“配置文件”。而在工作区的底部，会出现执行训练按钮。与左侧的三个功能按钮一样，当鼠标悬停在“执行训练”按钮上时，按钮会变色同时光标会变成手指。

在训练过程中，每一轮训练的结果都将被输出到软件下方的“日志输出”窗口以便用户了解当前的训练情况。

当训练结束后，软件会记录训练过程中测试集的准确率和 Loss 值，并将其在右侧窗口进行可视化。



当训练和测试结束后，用户可以选取预测功能，使用刚才自己训练的模型来对自己选取图像进行预测。

如下图所示，用户选取 MNIST 数据中某一个图像，该图像通过肉眼观察为数字“9”，

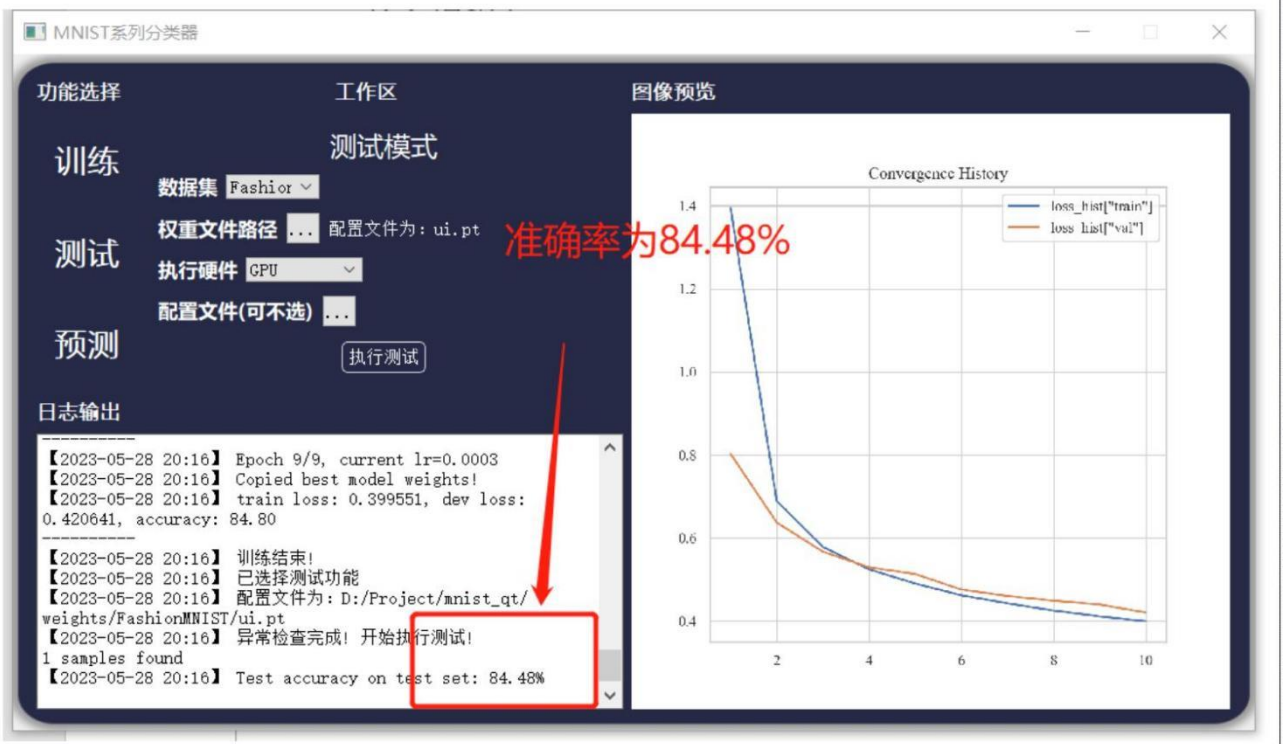
此时软件输出的预测结果为“9”，预测正确。

本软件除了可以在 MNIST 数据集上进行预测外，还可以在 FashionMNIST 数据集上进行预测。

如下图所示，用户选取了 FashionMNIST 数据集中的一张图片，肉眼可见该图像为“汗衫”，软件识别的结果为“汗衫”，识别正确！

本软件准确性高，在默认参数下，在 MNIST 数据集默认的测试集下的准确率可以达到 97.22%。

在 FashionMNIST 数据集下的准确率可以达到 84.48%。



接下来介绍深度学习工程部分。

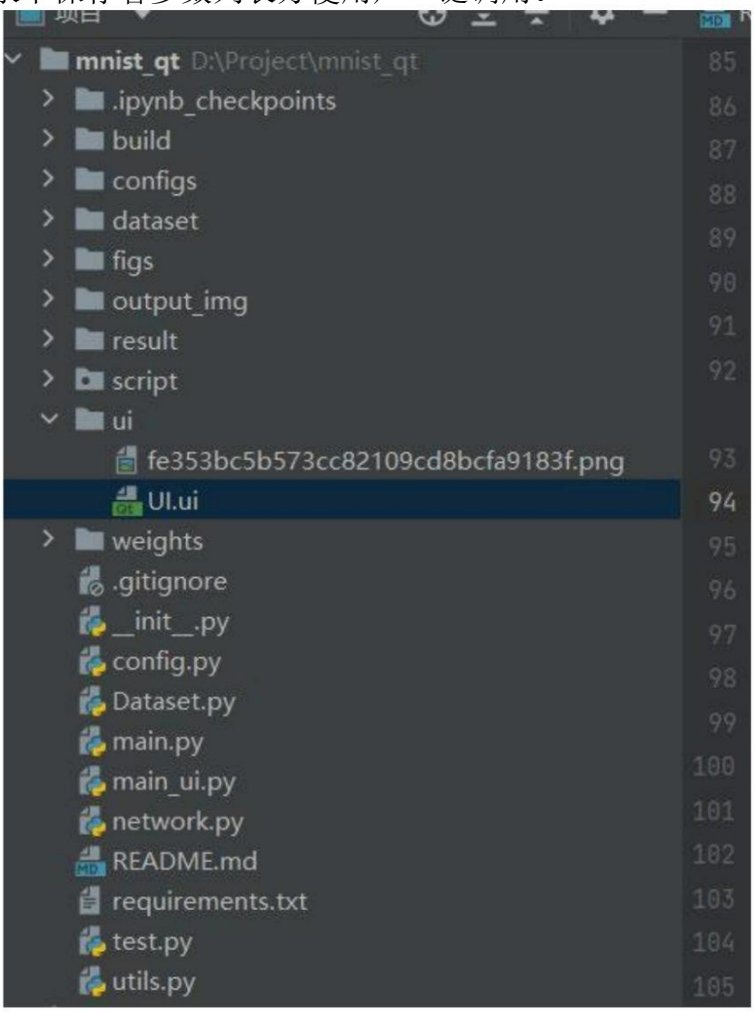
本软件的目录严格按照工程软件标准设计主要分为如下几个文件。

- main_ui.py 图形界面的主程序，其调用 main.py 文件中的函数。
- main.py 逻辑部分的主程序，软件的大部分逻辑函数都在该文件中实现。
- network.py 神经网络的结构在此文件中定义。
- utils.py 该文件中实现了各种工具类函数，方便其他文件进行调用。主要包含如下几个

功能。

- 根据训练过程中的准确率和 loss 值绘图。
- 绘制热度图
- 读取图像数据并将其转为 Numpy 数组
- 自动调参

- requirements.txt 该文件可以帮助用户快速的部署环境。
- README.md 该文件用于介绍该项目。
- ./ui 该目录下保存着此软件的 ui 文件
- ./script 该目录下保存着实现该项目的脚本文件
  - get_mnist.py 该文件用于下载 MNIST 和 FashionMNIST 数据集、解压、生成文件列表、整理文件的格式以便于软件的读取。
- ./result 该目录下保存着用户训练得到的权重文件
- ./output_img 该目录下保存着软件生成的所有的图片和临时图片。
- ./figs 该目录下保存着用户生成的可视化图片。例如记录每一轮训练后模型的准确率和 loss 值的图片。
- ./dataset 该目录下保存着数据集。分别是 MNIST 和 FashionMNIST 数据集。
- ./configs 该目录下保存着参数列表方便用户一键调用。



当开启训练任务时，程序先使用 yacs 库加载默认参数。
默认参数如下。

```
1. _C = CN()
```

```
_C.BASE = ['']

_C.DATA = CN()
_C.DATA.name = 'MNIST'
_C.DATA.shape_in_c = 1
_C.DATA.shape_in_h = 28
_C.DATA.shape_in_w = 28
_C.DATA.num_classes = 10
_C.DATA.data_path = './dataset/'
_C.DATA.dataset_list_train = './dataset/train_{}.txt'
_C.DATA.dataset_list_test = './dataset/test_{}.txt'

_C.MODEL = CN()
_C.MODEL.initial_filters = 8
_C.MODEL.num_fc = 10
_C.MODEL.dropout_rate = 0.2
_C.MODEL.weight_path = "weights/weights.pt"

_C.TRAIN = CN()
_C.TRAIN.OPTIMIZER = CN()
_C.TRAIN.OPTIMIZER.NAME = 'Adam'
_C.TRAIN.epoch_num = 10
_C.TRAIN.batch_size = 512
_C.TRAIN.lr = 0.01

_C.TEST = CN()
_C.TEST.batch_size = 1500

_C.IosForest = CN()
# _C.IosForest.processing = 'ori'
# _C.IosForest.processing = 'PCA'
_C.IosForest.processing = 'PCA'
_C.IosForest.dataset_path = './dataset/{}/raw/test'
_C.IosForest.n_estimators = 100
_C.IosForest.contamination = 0.08
_C.IosForest.sensitive_points_save_path = './SensitivePoints/sp'

_C.attack = CN()
_C.attack.pixel_num = 5
_C.attack.beta = 1
_C.attack.neighbor_size = 5
_C.attack.save_fig_max_num = 0
_C.attack.save_fig_path = './output_img/{}/'

```

```
_C.DE = CN()
_C.DE.step = 10
_C.DE.popsize = 10

_C.auto = CN()
_C.auto.auto = False
_C.auto.csv_path = 'result/{}/auto_'
_C.auto.feature_list = ['tsfel', 'PCA']
# _C.auto.beta_list = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
# _C.auto.pixel_num_list = [1, 3, 5, 10, 20]

_C.auto.beta_list = [0.1]
_C.auto.pixel_num_list = [1]
```

然后，根据用户选择的参数进行参数更新。

```
def main_training(dataset_name, output_folder_path, Device, yaml_filename, textBrowser):
    # print(type(textBrowser))
    global device
    if Device == 'GPU':
        device = torch.device("cuda")
    elif Device == 'CPU':
        device = torch.device("cpu")
    else:
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    global config
    if os.path.exists(yaml_filename):
        config = get_config(yaml_filename)
    else:
        config = get_config('configs/configMNIST.yaml')

    # 定义模型参数
    params_model = {
        "shape_in": (config.DATA.shape_in_c, config.DATA.shape_in_h, config.DATA.shape_in_w),
        "initial_filters": config.MODEL.initial_filters,
        "num_fc1": config.MODEL.num_fc,
        "dropout_rate": config.MODEL.dropout_rate,
        "num_classes": config.TRAIN.epoch_num,
        "weight_path": config.MODEL.weight_path
    }

    params_train = {
        "train": None,
```

```
        "val": None,
        "epochs": config.TRAIN.epoch_num,
        "optimiser": None,
        "lr_change": None,
        "f_loss": nn.NLLLoss(reduction="sum"),
        "weight_path": output_folder_path + '/' + 'ui.pt',
    }
    config.DATA.name = dataset_name
    train_dl, val_dl = get_dataset_dl(config=config)

    cnn_model = Network(config).to(device=device)
    if config.TRAIN.OPTIMIZER.NAME == 'Adam':
        opt = optim.Adam(cnn_model.parameters(), lr=config.TRAIN.lr)
    else:
        opt = optim.Adam(cnn_model.parameters(), lr=3e-4)
```

在定义网络结构的部分会使用 Network 函数，该函数来来自于文件 network.py，在该文件下定义了默认的网络结构。这是一个非常轻量级的网络，设计这个网络的初衷是为了用户能够快速训练。

值得一提的是，在网络的设计上，我实现了网络针对图像的大小进行动态的改变的功能，该功能主要使用下面这个函数实现，这个函数可以根据上一层输出的张量的形状动态定义下一层网络的参数。

```
def findConv2dOutShape(hin, win, conv, pool=2):
    """
    计算指定形状的输入经过卷积层后的输出形状
    """
    # get conv arguments
    kernel_size = conv.kernel_size
    stride = conv.stride
    padding = conv.padding
    dilation = conv.dilation

    hout = np.floor((hin + 2 * padding[0] - dilation[0] * (kernel_size[0] - 1) -
1) / stride[0] + 1)
    wout = np.floor((win + 2 * padding[1] - dilation[1] * (kernel_size[1] - 1) -
1) / stride[1] + 1)

    if pool:
        hout /= pool
        wout /= pool
    return int(hout), int(wout)
```

神经网络定义如下。

```
class Network(nn.Module):

    # Network Initialisation
```

```
    def __init__(self, config):
        """
         定义模型结构
        """
        super(Network, self).__init__()
        # 定义输入形状
        Cin, Hin, Win = config.DATA.shape_in_c, config.DATA.shape_in_h, config.DATA.shape_in_w
        # 定义卷积核个数
        init_f = config.MODEL.initial_filters
        # 定义全连接层中神经元个数
        num_fc1 = config.MODEL.num_fc
        # 定义分类数
        num_classes = config.DATA.num_classes
        # 定义 dropout率
        self.dropout_rate = config.MODEL.dropout_rate

        # Convolution Layers
        self.conv1 = nn.Conv2d(Cin, init_f, kernel_size=3)
        h, w = findConv2dOutShape(Hin, Win, self.conv1)
        self.conv2 = nn.Conv2d(init_f, 2 * init_f, kernel_size=3)
        h, w = findConv2dOutShape(h, w, self.conv2)
        # self.conv3 = nn.Conv2d(2 * init_f, 4 * init_f, kernel_size=3)
        # h, w = findConv2dOutShape(h, w, self.conv3)
        # self.conv4 = nn.Conv2d(4 * init_f, 8 * init_f, kernel_size=3)
        # h, w = findConv2dOutShape(h, w, self.conv4)

        # compute the flatten size
        self.num_flatten = h * w * 2 * init_f
        # 定义线性变换层
        self.fc1 = nn.Linear(self.num_flatten, num_fc1)
        self.fc2 = nn.Linear(num_fc1, num_classes)

    def forward(self, X):
        X = F.relu(self.conv1(X))
        X = F.max_pool2d(X, 2, 2)
        X = F.relu(self.conv2(X))
        X = F.max_pool2d(X, 2, 2)
        # X = F.relu(self.conv3(X))
        # X = F.max_pool2d(X, 2, 2)
        # X = F.relu(self.conv4(X))
        # X = F.max_pool2d(X, 2, 2)
        X = X.view(-1, self.num_flatten)
        # 定义激活函数
```

```
        X = F.relu(self.fc1(X))
        # 定义 dropout函数
        X = F.dropout(X, self.dropout_rate)
        X = self.fc2(X)
        return F.log_softmax(X, dim=1)
```

在定义数据集部分会调用 get_dataset_dl 函数，该函数来自于 Dataset.py 文件。该函数首先会判断输入的数据集名称是否合法。合法的数据集字符为“MNIST”和“FashionMNIST”，如果不合法就会升起一个报错“raise Exception("Dataset does not exist") ”。如果输入的是合法字符，就会去目录中加载对应的数据集，并返回一个 dataloader 类型的实例。

```
def get_dataset_dl(config):
    if not config.auto.auto:
        print('Doing {}'.format(config.DATA.name))
    if config.DATA.name == 'MNIST':
        train_loader = torch.utils.data.DataLoader(
            torchvision.datasets.MNIST(config.DATA.data_path, train=True, downloa
d=False,
                                       transform=torchvision.transforms.Compose([

                                           torchvision.transforms.ToTensor(),
                                           torchvision.transforms.Normalize(
                                               (0.1307,), (0.3081,))
                                       ])),
            batch_size=config.TRAIN.batch_size, shuffle=True, pin_memory=True)
        test_loader = torch.utils.data.DataLoader(
            torchvision.datasets.MNIST(config.DATA.data_path, train=False, downlo
ad=False,
                                       transform=torchvision.transforms.Compose([

                                           torchvision.transforms.ToTensor(),
                                           torchvision.transforms.Normalize(
                                               (0.1307,), (0.3081,))
                                       ])),
            batch_size=config.TEST.batch_size, shuffle=True, pin_memory=True)
        return train_loader, test_loader
    elif config.DATA.name == 'FashionMNIST':
        train_loader = torch.utils.data.DataLoader(
            torchvision.datasets.FashionMNIST(config.DATA.data_path, train=True,
download=False,
                                              transform=torchvision.transforms.Co
mpose([
                                                  torchvision.transforms.ToTensor
(),
                                                  torchvision.transforms.Normaliz
e(
```

```
                                                    (0.1307,), (0.3081,))
                                                ])),
            batch_size=config.TRAIN.batch_size, shuffle=True, pin_memory=True)
        test_loader = torch.utils.data.DataLoader(
            torchvision.datasets.FashionMNIST(config.DATA.data_path, train=False,
 download=False,
                                                transform=torchvision.transforms.Co
mpose([
                                                    torchvision.transforms.ToTensor
(),
                                                    torchvision.transforms.Normaliz
e(
                                                    (0.1307,), (0.3081,))
                                                ])),
            batch_size=config.TEST.batch_size, shuffle=True, pin_memory=True)
        return train_loader, test_loader
    else:
        raise Exception("Dataset does not exist")
```

然后定义优化器、损失函数。

```
params_train['train'] = train_dl
params_train['val'] = val_dl
params_train['optimiser'] = opt
params_train['lr_change'] = ReduceLROnPlateau(opt,
                                              mode='min',
                                              factor=0.5,
                                              patience=20,
                                              verbose=False)
```

使用 summary 函数可视化网络结构。

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 8, 26, 26]              80
            Conv2d-2           [-1, 16, 11, 11]           1,168
            Linear-3                 [-1, 100]          40,100
            Linear-4                  [-1, 10]           1,010
================================================================
```

输入训练子函数进行训练。

```
def train_val(model, params, verbose=True, textBrowser=None):
    # Get the parameters
    epochs = params["epochs"]
    loss_func = params["f_loss"]
    opt = params["optimiser"]
    train_dl = params["train"]
    val_dl = params["val"]
```

```
    lr_scheduler = params["lr_change"]
    weight_path = params["weight_path"]

    # loss_history和 metric_history用于绘图
    loss_history = {"train": [], "val": []}
    metric_history = {"train": [], "val": []}
    best_model_wts = copy.deepcopy(model.state_dict())
    best_loss = float('inf')
    ''''' Train Model n_epochs '''
    for epoch in range(epochs):

        ''''' Get the Learning Rate '''
        current_lr = get_lr(opt)
        if verbose:
            ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
            print(ttt + 'Epoch {}/{}, current lr={}'.format(epoch, epochs - 1, current_lr))
            textBrowser.append(ttt + 'Epoch {}/{}, current lr={}'.format(epoch, epochs - 1, current_lr))
            # 设置滚动条到最低部
            textBrowser.ensureCursorVisible()  # 游标可用
            cursor = textBrowser.textCursor()  # 设置游标
            pos = len(textBrowser.toPlainText())  # 获取文本尾部的位置
            cursor.setPosition(pos)  # 游标位置设置为尾部
            textBrowser.setTextCursor(cursor)  # 滚动到游标位置

        '''''

        Train Model Process

        '''

        model.train()
        train_loss, train_metric = loss_epoch(model, loss_func, train_dl, opt)

        # collect losses
        loss_history["train"].append(train_loss)
        metric_history["train"].append(train_metric)

        '''''

        Evaluate Model Process

        '''
```

```
50.
51.         model.eval()
52.         with torch.no_grad():
53.             val_loss, val_metric = loss_epoch(model, loss_func, val_dl)
54.
55.         # store best model
56.         if val_loss < best_loss:
57.             best_loss = val_loss
58.             best_model_wts = copy.deepcopy(model.state_dict())
59.
60.             # store weights into a local file
61.             torch.save(model.state_dict(), weight_path.format(config.DATA.name))
62.             if verbose:
63.                 ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
64.                 print(ttt + "Copied best model weights!")
65.                 textBrowser.append(ttt + "Copied best model weights!")
66.                 # 设置滚动条到最低部
67.                 textBrowser.ensureCursorVisible()  # 游标可用
68.                 cursor = textBrowser.textCursor()  # 设置游标
69.                 pos = len(textBrowser.toPlainText())  # 获取文本尾部的位置
70.                 cursor.setPosition(pos)  # 游标位置设置为尾部
71.                 textBrowser.setTextCursor(cursor)  # 滚动到游标位置
72.
73.         # collect loss and metric for validation dataset
74.         loss_history["val"].append(val_loss)
75.         metric_history["val"].append(val_metric)
76.
77.         # learning rate schedule
78.         lr_scheduler.step(val_loss)
79.         if current_lr != get_lr(opt):
80.             if verbose:
81.                 ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
82.                 print(ttt + "Loading best model weights!")
83.                 textBrowser.append(ttt + "Loading best model weights!")
84.                 # 设置滚动条到最低部
85.                 textBrowser.ensureCursorVisible()  # 游标可用
86.                 cursor = textBrowser.textCursor()  # 设置游标
87.                 pos = len(textBrowser.toPlainText())  # 获取文本尾部的位置
88.                 cursor.setPosition(pos)  # 游标位置设置为尾部
89.                 textBrowser.setTextCursor(cursor)  # 滚动到游标位置
90.             model.load_state_dict(best_model_wts)
91.
92.         if verbose:
```

```
            ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
            print(ttt + f"train loss: {train_loss:.6f}, dev loss: {val_loss:.6f}, accuracy: {100 * val_metric:.2f}")
            textBrowser.append(
                ttt + f"train loss: {train_loss:.6f}, dev loss: {val_loss:.6f}, accuracy: {100 * val_metric:.2f}")
            # 设置滚动条到最低部
            textBrowser.ensureCursorVisible()  # 游标可用
            cursor = textBrowser.textCursor()  # 设置游标
               pos = len(textBrowser.toPlainText())  # 获取文本尾部的位置
               cursor.setPosition(pos)  # 游标位置设置为尾部
               textBrowser.setTextCursor(cursor)  # 滚动到游标位置
               print("-" * 10)
               textBrowser.append("-" * 10)
               # 设置滚动条到最低部
               textBrowser.ensureCursorVisible()  # 游标可用
               cursor = textBrowser.textCursor()  # 设置游标
               pos = len(textBrowser.toPlainText())  # 获取文本尾部的位置
               cursor.setPosition(pos)  # 游标位置设置为尾部
               textBrowser.setTextCursor(cursor)  # 滚动到游标位置

               # load best model weights
        model.load_state_dict(best_model_wts)

        return model, loss_history, metric_history
```

训练结束后，返回训练后的权重参数和训练日志。

```
cnn_model, loss_hist, metric_hist = train_val(cnn_model, params_train, textBrowser=textBrowser)

fig_hist(params=params_train, loss_hist=loss_hist, metric_hist=metric_hist, config=config)
```

将日志输出由 utils.py 文件提供的 fig_hist 功能实现训练过程中准确率和 loss 值的可视化。

```
def fig_hist(params, loss_hist, metric_hist, config=None):
    """
    绘制训练过程中 loss值和准确率的变化
    """
    epochs = params["epochs"]
    # (655, 525)
    fig, ax = plt.subplots(1, 1, figsize=(6, 6))

    sns.lineplot(x=[*range(1, epochs + 1)], y=loss_hist["train"], ax=ax, label='loss_hist["train"]')
```

```
    sns.lineplot(x=[*range(1, epochs + 1)], y=loss_hist["val"], ax=ax, label='los
s_hist["val"]')
    plt.title('Convergence History')

    plt.savefig('figs/log.png', dpi=200)
```

在 utils.py 文件的开头需要定义字体，否则会出现中文显示异常的问题。

```
config = {
    "font.family": 'serif',
    "mathtext.fontset": 'stix',
    "font.serif": ['Times New Roman'],
}
from matplotlib import rcParams

rcParams.update(config)
sns.set(style='whitegrid', font='Serif')
```

当用户选用测试功能时，首先和训练功能一样，先加载默认参数，然后进入到测试函数中。

```
def main_testing(dataset_name, pth_path, Device, yaml_filename, textBrowser):
    global device
    if Device == 'GPU':
        device = torch.device("cuda")
    elif Device == 'CPU':
        device = torch.device("cpu")
    else:
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    global config
    if os.path.exists(yaml_filename):
        config = get_config(yaml_filename)
    else:
        config = get_config('configs/configMNIST.yaml')

    config.DATA.name = dataset_name
    # 定义模型参数
    params_model = {
        "shape_in": (config.DATA.shape_in_c, config.DATA.shape_in_h, config.DATA.
shape_in_w),
        "initial_filters": config.MODEL.initial_filters,
        "num_fc1": config.MODEL.num_fc,
        "dropout_rate": config.MODEL.dropout_rate,
        "num_classes": config.TRAIN.epoch_num,
        "weight_path": config.MODEL.weight_path if pth_path == None else pth_path

    }
```

```
    train_set, val_set = get_dataset_dl(config=config)
    ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
    print(ttt, len(val_set), 'samples found')
    textBrowser.append(str(len(val_set)) + ' samples found')
    # 设置滚动条到最低部
    textBrowser.ensureCursorVisible()  # 游标可用
    cursor = textBrowser.textCursor()  # 设置游标
    pos = len(textBrowser.toPlainText())  # 获取文本尾部的位置
    cursor.setPosition(pos)  # 游标位置设置为尾部
    textBrowser.setTextCursor(cursor)  # 滚动到游标位置
    cnn_model = Network(config).to(device=device)
    cnn_model.load_state_dict(torch.load(params_model["weight_path"]))
    acc = inference(model=cnn_model, dataset_dl=val_set, device=device, config=co
nfig)
    ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
    print(ttt + 'Test accuracy on test set: %0.2f%%\n' % acc)
    textBrowser.append(ttt + 'Test accuracy on test set: %0.2f%%\n' % acc)
    # 设置滚动条到最低部
    textBrowser.ensureCursorVisible()  # 游标可用
    cursor = textBrowser.textCursor()  # 设置游标
    pos = len(textBrowser.toPlainText())  # 获取文本尾部的位置
    cursor.setPosition(pos)  # 游标位置设置为尾部
    textBrowser.setTextCursor(cursor)  # 滚动到游标位置
```

注意，在测试功能中，我用了很大的篇幅将测试功能的输出同步输出到软件的日志输出框中。为了增强用户的交互性体验，我在每次输出日志时都会打印当前的时间。

```
ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
print(ttt, len(val_set), 'samples found')
```

当用户选取对 FashionMNIST 数据集进行预测时，会进入到 predict_img 函数中。

```
def predict_img(img_path, weight_path, textBrowser=None):
    is_fashion = True if "Fashion" in weight_path else False
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    config = get_config('configs/configMNIST.yaml')
    model = Network(config).to(device=device)
    model.load_state_dict(torch.load(weight_path))

    # 加载输入图像
    # image_path = 'path/to/your/image.jpg'
    image = Image.open(img_path)
    # 预处理输入图像
    image_tensor = transform(image)
    # image_tensor = image_tensor.unsqueeze(0)

    # 将模型设置为评估模式
    model.eval()
```

```
17.
18.     # 使用模型进行预测
19.     with torch.no_grad():
20.         output = model(image_tensor.to(device=device))
21.
22.     # 获取预测结果
23.     _, predicted = torch.max(output, 1)
24.
25.     res = predicted.item()
26.
27.     ls = ["T恤", "裤子", "套衫", "裙子", "外套", "凉鞋", "汗衫", "运动鞋", "包", "踝
    靴"]
28.     if is_fashion:
29.         res = ls[predicted.item()]
30.     ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
31.     textBrowser.append(ttt + "预测结果为:{}".format(res))
32.     # 设置滚动条到最低部
33.     textBrowser.ensureCursorVisible()  # 游标可用
34.     cursor = textBrowser.textCursor()  # 设置游标
35.     pos = len(textBrowser.toPlainText())  # 获取文本尾部的位置
36.     cursor.setPosition(pos)  # 游标位置设置为尾部
37.     textBrowser.setTextCursor(cursor)  # 滚动到游标位置
38.     # 打印预测结果
39.     ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
40.     print(ttt + predicted.item())
```

该函数会将模型输出的数字转换为用户所能接受的字符。例如当模型输出的类别为 0 时，该函数会将 0 转变为“T 恤”。再比如当模型输出“2”时，该函数会将 2 转变为“套衫”。

在 main_ui.py 中，我通过调用上述函数实现 GUI 界面中用户所需要的功能。

```
1.  import os.path
2.  import time
3.  from PySide2.QtGui import QPixmap
4.  from PySide2.QtGui import QFont
5.  from PySide2.QtUiTools import QUiLoader
6.  from os.path import exists, basename
7.  from PySide2.QtWidgets import QApplication, QWidget, QPushButton, QLabel, QFileDi
    alog, QVBoxLayout, QComboBox
8.  from PySide2.QtWidgets import QHBoxLayout
9.  from PySide2.QtGui import QColor, QPalette
10. from main import main_training, main_testing, predict_img
11. # from script.get_mnist import get_list
12. from PySide2.QtCore import Qt, QThread, Signal
13.
14. # 创建调色板对象
```

```
palette = QPalette()

# 创建颜色对象
color = QColor(255, 255, 255)  # 白色

# 设置标签的字体颜色
palette.setColor(QPalette.WindowText, color)

# 创建一个 QFont对象并设置字体、大小和加粗属性
font_ = QFont('Microsoft YaHei', 10, QFont.Bold)  # 这里使用微软雅黑字体，字号为12，加粗属性为 True

class BackendThread_train(QThread):

    def __init__(self, dataset_name, output_folder_path, device, yaml_filename,
                 textBrowser, image_label, able):
        QThread.__init__(self)
        self.dataset_name = dataset_name
        self.output_folder_path = output_folder_path
        self.device = device
        self.yaml_filename = yaml_filename
        self.textBrowser = textBrowser
        self.image_label = image_label
        self.able = able

    def run(self):
        main_training(self.dataset_name, self.output_folder_path, self.device, self.yaml_filename,
                      textBrowser=self.textBrowser)
        pixmap = QPixmap('figs/log.png').scaled(504, 506)
        self.image_label.setPixmap(pixmap)
        ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
        print(ttt + "训练结束!")
        self.textBrowser.append(ttt + "训练结束!")
        self.able(True)


class BackendThread_test(QThread):

    def __init__(self, dataset_name, output_folder_path, device, yaml_filename,
                 textBrowser, able):
        QThread.__init__(self)
        self.dataset_name = dataset_name
        self.output_folder_path = output_folder_path
```

```
57.         self.device = device
58.         self.yaml_filename = yaml_filename
59.         self.textBrowser = textBrowser
60.         self.able = able
61.
62.     def run(self):
63.         # ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
64.         # self.textBrowser.append(ttt + '文件列表已生成!')
65.         main_testing(self.dataset_name, self.output_folder_path, self.device, sel
    f.yaml_filename,
66.                      textBrowser=self.textBrowser)
67.         ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
68.         print(ttt + "测试结束!")
69.         self.textBrowser.append(ttt + "测试结束!")
70.         self.able(True)
71.
72.
73. class BackendThread_predict(QThread):
74.
75.     def __init__(self, img_path, weight_path, textBrowser, image_label):
76.         QThread.__init__(self)
77.         self.img_path = img_path
78.         self.weight_path = weight_path
79.         self.textBrowser = textBrowser
80.         self.image_label = image_label
81.
82.     def run(self):
83.         pixmap = QPixmap(self.img_path).scaled(600, 600)
84.         self.image_label.setPixmap(pixmap)
85.
86.         predict_img(self.img_path, self.weight_path, textBrowser=self.textBrowser
    )
87.         ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
88.         print(ttt + "预测结束!")
89.         self.textBrowser.append(ttt + "预测结束!")
90.         # self.able(True)
91.
92.
93. class Stats:
94.     def __init__(self):
95.         self.pt_filename = ""
96.         self.yaml_filename = ""
97.         self.fig_filename = ""
98.         self.output_folder_path = ""
```

```
99.         self.ui = QUiLoader().load('ui/UI.ui')
100.           # self.ui.setWindowFlag(Qt.FramelessWindowHint)
101.
102.           self.ui.button_train.clicked.connect(self.train)
103.           self.ui.button_test.clicked.connect(self.test)
104.           self.ui.button_predict.clicked.connect(self.predict)
105.           self.ui.textBrowser.verticalScrollBar().setValue(self.ui.textBrowser.v
    erticalScrollBar().maximum())
106.
107.           self.page_0()
108.           self.page_1()
109.           self.page_2()
110.           self.page_3()
111.
112.           self.page1.setVisible(False)
113.           self.page2.setVisible(False)
114.           self.page3.setVisible(False)
115.
116.           layout = self.ui.verticalLayout
117.           layout.addWidget(self.page0)
118.           self.page0.setVisible(True)
119.           layout.addWidget(self.page1)
120.           layout.addWidget(self.page2)
121.           layout.addWidget(self.page3)
122.
123.           ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
124.           self.ui.textBrowser.append(ttt + 'MNIST分类器启动成功。')
125.
126.       def page_0(self):
127.           self.page0 = QWidget()
128.           self.label0 = QLabel(
129.               "|" + "欢迎使用 MNIST分类
    器!".center(20) + "|" + "\n" + "      |" + "作者：张子扬 ".center(20) + "|")
130.           font = QFont("微软雅黑", 12)  # 创建一个 12号微软雅黑字体
131.           self.label0.setPalette(palette)
132.           self.label0.setFont(font)  # 将 QLabel的字体设置为微软雅黑
133.           layout0 = QHBoxLayout(self.page0)
134.           layout0.addWidget(self.label0, alignment=Qt.AlignCenter)
135.
136.       def page_1(self):
137.           self.page1 = QWidget()
138.
139.           layout_0 = QVBoxLayout(self.page1)
140.           self.label0 = QLabel("训练模式")
```

```
141.            font = QFont("微软雅黑", 14)  # 创建一个 12号微软雅黑字体
142.            self.label0.setPalette(palette)
143.            self.label0.setFont(font)
144.            layout_0.addWidget(self.label0, alignment=Qt.AlignCenter)
145.
146.            layout_1 = QHBoxLayout(alignment=Qt.AlignLeft)
147.            self.label_1 = QLabel("数据集")
148.            self.label_1.setFont(font_)
149.            self.label_1.setPalette(palette)
150.            self.combo_box_1 = QComboBox()
151.            self.combo_box_1.currentIndexChanged.connect(self.get_dataset)
152.            self.combo_box_1.setFixedWidth(130)
153.            self.combo_box_1.addItem("MNIST")
154.            self.combo_box_1.addItem("FashionMNIST")
155.            layout_1.addWidget(self.label_1)
156.            layout_1.addWidget(self.combo_box_1)
157.
158.            layout_2 = QHBoxLayout(alignment=Qt.AlignLeft)
159.            self.folder_label_2_0 = QLabel('权重文件输出路径')
160.            self.folder_label_2_0.setFont(font_)
161.            self.folder_label_2_0.setPalette(palette)
162.            self.folder_label_2 = QLabel()
163.            self.folder_label_2.setPalette(palette)
164.            self.folder_button_2 = QPushButton('...')
165.            self.folder_button_2.setFixedWidth(30)
166.            self.folder_button_2.clicked.connect(self.choose_output_folder)
167.            layout_2.addWidget(self.folder_label_2_0)
168.            layout_2.addWidget(self.folder_button_2)
169.            layout_2.addWidget(self.folder_label_2)
170.
171.            layout_3 = QHBoxLayout(alignment=Qt.AlignCenter)
172.            self.folder_label_3 = QLabel()
173.            self.train_button = QPushButton('执行训练')
174.            self.train_button.setFont(font_)
175.            self.train_button.setCursor(Qt.PointingHandCursor)
176.            self.train_button.setStyleSheet('''''
177.                    QPushButton {
178.                        color: white;
179.                        background-color: (39,42,71);
180.                        border: 1px solid #dcdfe6;
181.                        padding: 5px;
182.                        border-radius: 5px;
183.                    }
184.
```

```
                    QPushButton:hover {
                        background-color: #ecf5ff;
                        color: #409eff;
                    }

                    QPushButton:pressed, QPushButton:checked {
                        border: 1px solid #3a8ee6;
                        color: #409eff;
                    }
                    ''')

        self.train_button.setPalette(palette)
        self.train_button.clicked.connect(self.train_exe)
        layout_3.addWidget(self.train_button)

        layout_4 = QHBoxLayout(alignment=Qt.AlignLeft)
        self.label_4 = QLabel("执行硬件")
        self.label_4.setPalette(palette)
        self.label_4.setFont(font_)
        self.combo_box = QComboBox()
        self.combo_box.currentIndexChanged.connect(self.get_device)
        self.combo_box.setFixedWidth(130)
        self.combo_box.addItem("自动选择")
        self.combo_box.addItem("GPU")
        self.combo_box.addItem("CPU")
        layout_4.addWidget(self.label_4)
        layout_4.addWidget(self.combo_box)

        layout_5 = QHBoxLayout(alignment=Qt.AlignLeft)
        self.folder_label_5_0 = QLabel('配置文件(可不选)')
        self.folder_label_5_0.setPalette(palette)
        self.folder_label_5_0.setFont(font_)
        self.folder_label_5 = QLabel()
        self.folder_label_5.setPalette(palette)
        self.folder_button_5 = QPushButton('...')
        self.folder_button_5.setFixedWidth(30)
        self.folder_button_5.clicked.connect(self.select_yaml)
        layout_5.addWidget(self.folder_label_5_0)
        layout_5.addWidget(self.folder_button_5)
        layout_5.addWidget(self.folder_label_5)

        layout_0.addLayout(layout_1)
        layout_0.addLayout(layout_2)
        layout_0.addLayout(layout_4)
```

```
            layout_0.addLayout(layout_5)
            layout_0.addLayout(layout_3)

        def get_dataset(self):
            self.dataset_name = self.combo_box_1.currentText()

        def get_device(self):
            self.device = self.combo_box.currentText()

        def train_exe(self):

            if os.path.exists('figs/log.png'):
                os.remove('figs/log.png')

            if self.dataset_name == "":
                ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
                self.ui.textBrowser.append(ttt + '数据集名称为空,请检查!')
                return
            if self.output_folder_path == "":
                ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
                self.ui.textBrowser.append(ttt + '权重文件输出路径为空,请检查!')
                return
            if self.device == "":
                self.device = '自动选择'
            ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
            self.ui.textBrowser.append(ttt + '异常检查完成! 开始执行训练!')

            self.able_train(False)

            # print(type(self.ui.textBrowser))

            self.backend_thread_train = BackendThread_train(self.dataset_name, sel
    f.output_folder_path, self.device,
                                                            self.yaml_filename,
                                                            textBrowser=self.ui.te
    xtBrowser,
                                                            image_label=self.ui.im
    age_label, able=self.able_train)
            self.backend_thread_train.start()

        def able_train(self, flag):
            if flag == False:
                self.train_button.setEnabled(False)
                self.combo_box_1.setEnabled(False)
```

```
270.                 self.folder_button_2.setEnabled(False)
271.                 self.combo_box.setEnabled(False)
272.                 self.folder_button_5.setEnabled(False)
273.                 self.ui.button_train.setEnabled(False)
274.                 self.ui.button_test.setEnabled(False)
275.                 self.ui.button_predict.setEnabled(False)
276.             else:
277.                 self.train_button.setEnabled(True)
278.                 self.combo_box_1.setEnabled(True)
279.                 self.folder_button_2.setEnabled(True)
280.                 self.combo_box.setEnabled(True)
281.                 self.folder_button_5.setEnabled(True)
282.                 self.ui.button_train.setEnabled(True)
283.                 self.ui.button_test.setEnabled(True)
284.                 self.ui.button_predict.setEnabled(True)
285.
286.         def select_yaml(self):
287.             # 弹出文件选择对话框，获取用户选择的文件路径
288.             self.yaml_filename, _ = QFileDialog.getOpenFileName(None, "选择文件
    ", "", "yaml files (*.yaml)")
289.             if self.yaml_filename:
290.                 yaml_name = basename(self.yaml_filename)
291.                 ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
292.                 self.ui.textBrowser.append(ttt + '配置文件为：
    {}'.format(self.yaml_filename))
293.                 self.folder_label_5.setText(f'配置文件为：{yaml_name}')
294.
295.         def choose_output_folder(self):
296.             self.output_folder_path = QFileDialog.getExistingDirectory(None, '选择
    文件夹')
297.             if self.output_folder_path:
298.                 folder_name = basename(self.output_folder_path)
299.                 ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
300.                 self.ui.textBrowser.append(ttt + '权重文件输出文件夹为：
    {}'.format(self.output_folder_path))
301.                 self.folder_label_2.setText(f'已选择文件夹：{folder_name}')
302.
303.         def select_pt(self):
304.             # 弹出文件选择对话框，获取用户选择的文件路径
305.             self.pt_filename, _ = QFileDialog.getOpenFileName(None, "选择文件
    ", "", "pt files (*.pt)")
306.             if self.pt_filename:
307.                 pt_name = basename(self.pt_filename)
308.                 ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
```

```
309.                 self.ui.textBrowser.append(ttt + '配置文件为：
    {}'.format(self.pt_filename))
310.                 self.folder_label_2_page2.setText(f'配置文件为：{pt_name}')
311.
312.         def select_pt_page3(self):
313.             # 弹出文件选择对话框，获取用户选择的文件路径
314.             self.pt_filename, _ = QFileDialog.getOpenFileName(None, "选择文件
    ", "", "pt files (*.pt)")
315.             if self.pt_filename:
316.                 pt_name = basename(self.pt_filename)
317.                 ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
318.                 self.ui.textBrowser.append(ttt + '配置文件为：
    {}'.format(self.pt_filename))
319.                 self.folder_label_4_page3.setText(f'配置文件为：{pt_name}')
320.
321.         def page_2(self):
322.             self.page2 = QWidget()
323.
324.             layout_0_page2 = QVBoxLayout(self.page2)
325.             self.label0_page2 = QLabel("测试模式")
326.             font = QFont("微软雅黑", 14)  # 创建一个 12号微软雅黑字体
327.             self.label0_page2.setPalette(palette)
328.             self.label0_page2.setFont(font)
329.             layout_0_page2.addWidget(self.label0_page2, alignment=Qt.AlignCenter)

330.
331.             layout_1_page2 = QHBoxLayout(alignment=Qt.AlignLeft)
332.             self.label_1_page2 = QLabel("数据集")
333.             self.label_1_page2.setPalette(palette)
334.             self.label_1_page2.setFont(font_)
335.             self.combo_box_dataset_page2 = QComboBox()
336.             self.combo_box_dataset_page2.currentIndexChanged.connect(self.get_data
    set)
337.             self.combo_box_dataset_page2.setFixedWidth(80)
338.             self.combo_box_dataset_page2.addItem("MNIST")
339.             self.combo_box_dataset_page2.addItem("FashionMNIST")
340.             layout_1_page2.addWidget(self.label_1_page2)
341.             layout_1_page2.addWidget(self.combo_box_dataset_page2)
342.
343.             layout_2_page2 = QHBoxLayout(alignment=Qt.AlignLeft)
344.             self.folder_label_2_0_page2 = QLabel('权重文件路径')
345.             self.folder_label_2_0_page2.setPalette(palette)
346.             self.folder_label_2_0_page2.setFont(font_)
347.             self.folder_label_2_page2 = QLabel()
```

```
348.            self.folder_label_2_page2.setPalette(palette)
349.            self.folder_button_2_page2 = QPushButton('...')
350.            self.folder_button_2_page2.setFixedWidth(30)
351.            self.folder_button_2_page2.clicked.connect(self.select_pt)
352.            layout_2_page2.addWidget(self.folder_label_2_0_page2)
353.            layout_2_page2.addWidget(self.folder_button_2_page2)
354.            layout_2_page2.addWidget(self.folder_label_2_page2)
355.
356.            layout_3_page2 = QHBoxLayout(alignment=Qt.AlignCenter)
357.            self.folder_label_3_page2 = QLabel()
358.            self.test_button = QPushButton('执行测试')
359.            self.test_button.setCursor(Qt.PointingHandCursor)
360.            self.test_button.setStyleSheet('''''
361.                QPushButton {
362.                color: white;
363.                background-color: (39,42,71);
364.                border: 1px solid #dcdfe6;
365.                padding: 5px;
366.                border-radius: 5px;
367.            }
368.
369.            QPushButton:hover {
370.                background-color: #ecf5ff;
371.                color: #409eff;
372.                cursor: pointer;
373.            }
374.
375.            QPushButton:pressed, QPushButton:checked {
376.                border: 1px solid #3a8ee6;
377.                color: #409eff;
378.            }
379.
380.            #button3 {
381.                border-radius: 20px;
382.            }
383.
384.            ''')
385.
386.            self.test_button.setPalette(palette)
387.            self.test_button.clicked.connect(self.test_exe)
388.            layout_3_page2.addWidget(self.test_button)
389.
390.            layout_4_page2 = QHBoxLayout(alignment=Qt.AlignLeft)
391.            self.label_4_page2 = QLabel("执行硬件")
```

```
392.            self.label_4_page2.setPalette(palette)
393.            self.label_4_page2.setFont(font_)
394.            self.combo_box_page2 = QComboBox()
395.            self.combo_box_page2.currentIndexChanged.connect(self.get_device)
396.            self.combo_box_page2.setFixedWidth(100)
397.            self.combo_box_page2.addItem("自动选择")
398.            self.combo_box_page2.addItem("GPU")
399.            self.combo_box_page2.addItem("CPU")
400.            layout_4_page2.addWidget(self.label_4_page2)
401.            layout_4_page2.addWidget(self.combo_box_page2)
402.
403.            layout_5_page2 = QHBoxLayout(alignment=Qt.AlignLeft)
404.            self.folder_label_5_0_page2 = QLabel('配置文件(可不选)')
405.            self.folder_label_5_0_page2.setPalette(palette)
406.            self.folder_label_5_0_page2.setFont(font_)
407.            self.combo_box_page2 = QComboBox()
408.            self.folder_label_5_page2 = QLabel()
409.            self.folder_label_5_page2.setPalette(palette)
410.            self.folder_button_5_page2 = QPushButton('...')
411.            self.folder_button_5_page2.setFixedWidth(30)
412.            self.folder_button_5_page2.clicked.connect(self.select_yaml)
413.            layout_5_page2.addWidget(self.folder_label_5_0_page2)
414.            layout_5_page2.addWidget(self.folder_button_5_page2)
415.            layout_5_page2.addWidget(self.folder_label_5_page2)
416.
417.            layout_0_page2.addLayout(layout_1_page2)
418.            layout_0_page2.addLayout(layout_2_page2)
419.            layout_0_page2.addLayout(layout_4_page2)
420.            layout_0_page2.addLayout(layout_5_page2)
421.            layout_0_page2.addLayout(layout_3_page2)
422.
423.        def able_test(self, flag):
424.            if flag == False:
425.                self.train_button.setEnabled(False)
426.                self.combo_box_dataset_page2.setEnabled(False)
427.                self.folder_button_2.setEnabled(False)
428.                self.combo_box_page2.setEnabled(False)
429.                self.folder_button_5.setEnabled(False)
430.                self.ui.button_train.setEnabled(False)
431.                self.ui.button_test.setEnabled(False)
432.                self.ui.button_predict.setEnabled(False)
433.            else:
434.                self.train_button.setEnabled(True)
435.                self.combo_box_dataset_page2.setEnabled(True)
```

```
436.                 self.folder_button_2.setEnabled(True)
437.                 self.combo_box_page2.setEnabled(True)
438.                 self.folder_button_5.setEnabled(True)
439.                 self.ui.button_train.setEnabled(True)
440.                 self.ui.button_test.setEnabled(True)
441.                 self.ui.button_predict.setEnabled(True)
442.
443.         def test_exe(self):
444.             if self.dataset_name == "":
445.                 ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
446.                 self.ui.textBrowser.append(ttt + '数据集名称空,请检查!')
447.                 return
448.             if self.pt_filename == "":
449.                 ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
450.                 self.ui.textBrowser.append(ttt + '权重文件路径为空,请检查!')
451.                 return
452.             if self.device == "":
453.                 self.device = '自动选择'
454.             ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
455.             self.ui.textBrowser.append(ttt + '异常检查完成! 开始执行测试!')
456.
457.             self.able_test(False)
458.
459.             self.backend_thread_train = BackendThread_test(self.dataset_name, self
    .pt_filename, self.device,
460.                                                            self.yaml_filename,
461.                                                            textBrowser=self.ui.tex
    tBrowser, able=self.able_test)
462.             self.backend_thread_train.start()
463.
464.         def page_3(self):
465.             # print("预测")
466.             self.page3 = QWidget()
467.
468.             layout_0_page3 = QVBoxLayout(self.page3)
469.             self.label0_page3 = QLabel("预测模式")
470.             font = QFont("微软雅黑", 14)  # 创建一个 12号微软雅黑字体
471.             self.label0_page3.setPalette(palette)
472.             self.label0_page3.setFont(font)
473.             layout_0_page3.addWidget(self.label0_page3, alignment=Qt.AlignCenter)
474.
475.             layout_2_page3 = QHBoxLayout(alignment=Qt.AlignLeft)
476.             self.folder_label_2_0_page3 = QLabel('预测图像路径')
```

```
477.            self.folder_label_2_0_page3.setPalette(palette)
478.            self.folder_label_2_0_page3.setFont(font_)
479.            self.folder_label_2_page3 = QLabel()
480.            self.folder_label_2_page3.setPalette(palette)
481.            self.folder_button_2_page3 = QPushButton('...')
482.            self.folder_button_2_page3.setFixedWidth(30)
483.            self.folder_button_2_page3.clicked.connect(self.select_fig)
484.            layout_2_page3.addWidget(self.folder_label_2_0_page3)
485.            layout_2_page3.addWidget(self.folder_button_2_page3)
486.            layout_2_page3.addWidget(self.folder_label_2_page3)
487.
488.            layout_4_page3 = QHBoxLayout(alignment=Qt.AlignLeft)
489.            self.folder_label_4_0_page3 = QLabel('权重文件路径')
490.            self.folder_label_4_0_page3.setPalette(palette)
491.            self.folder_label_4_0_page3.setFont(font_)
492.            self.folder_label_4_page3 = QLabel()
493.            self.folder_label_4_page3.setPalette(palette)
494.            self.folder_button_4_page3 = QPushButton('...')
495.            self.folder_button_4_page3.setFixedWidth(30)
496.            self.folder_button_4_page3.clicked.connect(self.select_pt_page3)
497.            layout_4_page3.addWidget(self.folder_label_4_0_page3)
498.            layout_4_page3.addWidget(self.folder_button_4_page3)
499.            layout_4_page3.addWidget(self.folder_label_4_page3)
500.
501.            layout_3_page3 = QHBoxLayout(alignment=Qt.AlignCenter)
502.            # self.folder_label_3_page3 = QLabel()
503.            self.predict_button = QPushButton('执行预测')
504.            self.predict_button.setCursor(Qt.PointingHandCursor)
505.            self.predict_button.setStyleSheet('''''
506.                QPushButton {
507.                color: white;
508.                background-color: (39,42,71);
509.                border: 1px solid #dcdfe6;
510.                padding: 5px;
511.                border-radius: 5px;
512.            }
513.
514.            QPushButton:hover {
515.                background-color: #ecf5ff;
516.                color: #409eff;
517.                cursor: pointer;
518.            }
519.
520.            QPushButton:pressed, QPushButton:checked {
```

```
521.                border: 1px solid #3a8ee6;
522.                color: #409eff;
523.            }
524.
525.            #button3 {
526.                border-radius: 20px;
527.            }
528.
529.            ''')
530.
531.            self.predict_button.setPalette(palette)
532.            self.predict_button.clicked.connect(self.predict_exe)
533.            layout_3_page3.addWidget(self.predict_button)
534.
535.            layout_5_page3 = QHBoxLayout(alignment=Qt.AlignLeft)
536.            self.label_5_page3 = QLabel("执行硬件")
537.            self.label_5_page3.setPalette(palette)
538.            self.label_5_page3.setFont(font_)
539.            self.combo_box_page3 = QComboBox()
540.            self.combo_box_page3.currentIndexChanged.connect(self.get_device)
541.            self.combo_box_page3.setFixedWidth(100)
542.            self.combo_box_page3.addItem("自动选择")
543.            self.combo_box_page3.addItem("GPU")
544.            self.combo_box_page3.addItem("CPU")
545.            layout_5_page3.addWidget(self.label_5_page3)
546.            layout_5_page3.addWidget(self.combo_box_page3)
547.
548.            layout_0_page3.addLayout(layout_2_page3)
549.            layout_0_page3.addLayout(layout_4_page3)
550.            layout_0_page3.addLayout(layout_5_page3)
551.            layout_0_page3.addLayout(layout_3_page3)
552.
553.        def select_fig(self):
554.            # 弹出文件选择对话框，获取用户选择的文件路径
555.            self.fig_filename, _ = QFileDialog.getOpenFileName(None, "选择文件
    ", "", "jpg files (*.jpg)")
556.            if self.fig_filename:
557.                fig_name = basename(self.fig_filename)
558.                ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
559.                self.ui.textBrowser.append(ttt + '配置文件为：
    {}'.format(self.fig_filename))
560.                self.folder_label_2_page3.setText(f'配置文件为：{fig_name}')
561.
562.        def predict_exe(self):
```

```
563.            print("预测")
564.            if self.fig_filename == "":
565.                ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
566.                self.ui.textBrowser.append(ttt + '图像路径为空,请检查!')
567.                return
568.            if self.pt_filename == "":
569.                ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
570.                self.ui.textBrowser.append(ttt + '权重文件路径为空,请检查!')
571.                return
572.            if self.device == "":
573.                self.device = '自动选择'
574.            ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
575.            self.ui.textBrowser.append(ttt + '异常检查完成! 开始执行测试!')
576.
577.            self.backend_thread_predict = BackendThread_predict(self.fig_filename,
    self.pt_filename, self.ui.textBrowser,
578.                                                                image_label=self.u
   i.image_label)
579.            self.backend_thread_predict.start()
580.
581.        def train(self):
582.
583.            ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
584.            self.ui.textBrowser.append(ttt + '已选择训练功能')
585.
586.            self.ui.textBrowser.ensureCursorVisible()  # 游标可用
587.            cursor = self.ui.textBrowser.textCursor()  # 设置游标
588.            pos = len(self.ui.textBrowser.toPlainText())  # 获取文本尾部的位置
589.            cursor.setPosition(pos)  # 游标位置设置为尾部
590.            self.ui.textBrowser.setTextCursor(cursor)  # 滚动到游标位置
591.
592.            self.page0.setVisible(False)
593.            self.page1.setVisible(True)
594.            self.page2.setVisible(False)
595.            self.page3.setVisible(False)
596.            print("train")
597.
598.        def test(self):
599.
600.            ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
601.            self.ui.textBrowser.append(ttt + '已选择测试功能')
602.
603.            self.ui.textBrowser.ensureCursorVisible()  # 游标可用
604.            cursor = self.ui.textBrowser.textCursor()  # 设置游标
```

```
605.            pos = len(self.ui.textBrowser.toPlainText())  # 获取文本尾部的位置
606.            cursor.setPosition(pos)  # 游标位置设置为尾部
607.            self.ui.textBrowser.setTextCursor(cursor)  # 滚动到游标位置
608.
609.            self.page0.setVisible(False)
610.            self.page1.setVisible(False)
611.            self.page2.setVisible(True)
612.            self.page3.setVisible(False)
613.            print("test")
614.
615.        def predict(self):
616.
617.            ttt = time.strftime("【%Y-%m-%d %H:%M】 ", time.localtime())
618.            self.ui.textBrowser.append(ttt + '已选择预测功能')
619.
620.            self.ui.textBrowser.ensureCursorVisible()  # 游标可用
621.            cursor = self.ui.textBrowser.textCursor()  # 设置游标
622.            pos = len(self.ui.textBrowser.toPlainText())  # 获取文本尾部的位置
623.            cursor.setPosition(pos)  # 游标位置设置为尾部
624.            self.ui.textBrowser.setTextCursor(cursor)  # 滚动到游标位置
625.
626.            self.page0.setVisible(False)
627.            self.page1.setVisible(False)
628.            self.page2.setVisible(False)
629.            self.page3.setVisible(True)
630.            print("predict")
631.
632.
633.    app = QApplication([])
634.    stats = Stats()
635.    stats.ui.show()
636.    app.exec_()
```

实验中遇到的问题及解决办法：

在本次实验中，由于实验体量巨大，我遇到了很多的问题。

由于我有深度学习基础，因此在深度学习工程的构建上我并没有遇到太多的障碍。主要的障碍出现在 GUI 的设计上。

首次设计 GUI 界面对我来说是一个挑战，因为我自认为审美并不好，因此我参考了很多设计优美的软件，总结他们的设计经验。同时在 UI 的设计上我并不能熟练地使用一些功能。为此，我查阅了大量的资料，也看了很多的视频，最终设计了一个还算是优美的解决方法。

除此之外，多线程操作对我来说也是一个挑战，如果使用单线程的话，当软件在执行训练或者测试功能的时候，由于在子函数中停留的时间过长，系统会认为软件进入了无响应状态。此时就必须使用多线程操作，令主线程停留在前端界面，令子线程去执行背后的逻辑操作。事实证明，这种方法不仅可以提高软件的运转效率，在应对突发状况时也有显著的效

果。

当然，这个软件也存在一些问题，比如在进行迁移的时候背景照片有时候就找不到了，需要在 ui 文件中手动添加绝对路径。当时我认为这个问题是 pyqt5 源码的缺陷。

实验总结与心得体会：

在本次实验中，我设计了一个带有图形界面的 MNIST 分类器。在完成了老师要求的功能外，我还利用该软件实现了对 FashionMNIST 数据集的分类。我认为该软件的前端界面设计美观。后端界面设计先进。

软件严格按照工程要求完成，文件和函数的定义清晰明了，工程量巨大。粗略估计本工程的代码量达到了五千行。耗时一周完成。

本次实验对我来说意义重大，我认为最大的意义就是让我首次设计了有界面的软件。之前我设计的软件都是黑框界面，这样的软件缺少交互性也缺少了很多的趣味。

带图形界面的编程给我打开了新世界的大门，我发现图形界面的软件也没有那么难，虽然在设计的时候会比黑框界面麻烦很多，但是一旦设计完成，图形软件的易用性比黑框软件高了很多。

除此之外，在这次实验中，我还学习了多线程操作。虽然之前学习过多线程，但是之前仅仅是停留在理论阶段，如今自己动手实现还是令我非常有成就感的。