1 一、课设选题

（一）项目功能介绍

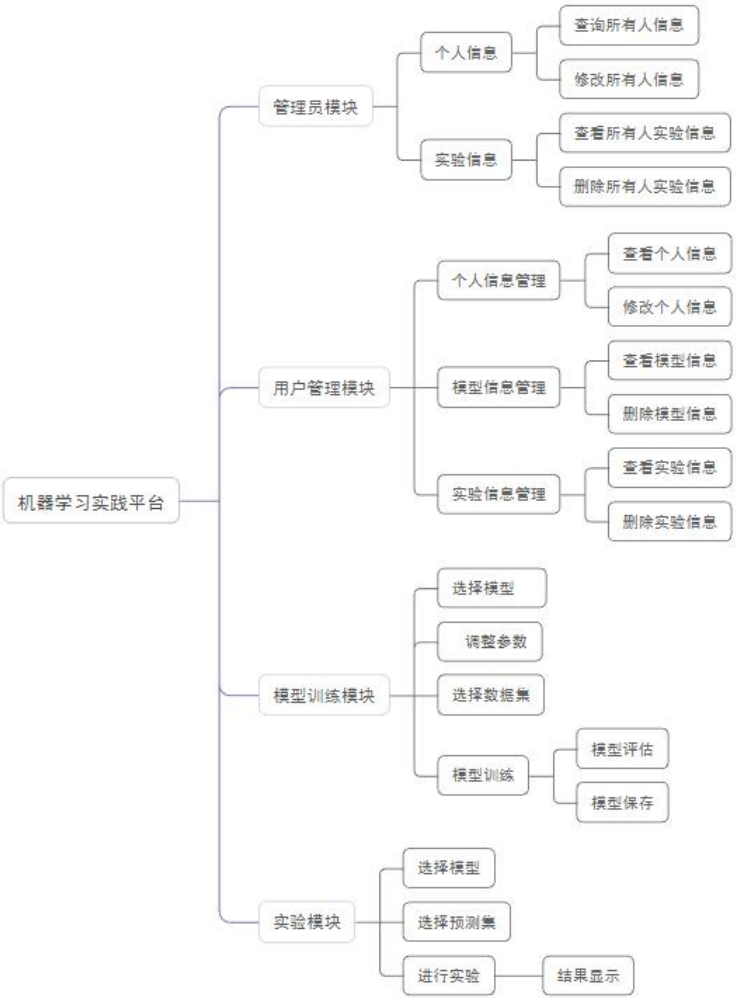

图 1 系统功能模块图

（二）选用的数据集

CIFAR-10 数据集

CIFAR-10 数据集由 10 个类别的 60000 个 32x32 彩色图像组成，每个类别有 6000 个图像。有 50000 张训练图像和 10000 张测试图像。

数据集分为 5 个训练批次和 1 个测试批次，每个批次有 10000 张图像。测试批次包含从每个类别中随机选择的 1000 张图像。训练批次以随机顺序

包含剩余的图像，但某些训练批次可能包含来自一个类的图像多于另一个类的图像。在它们之间，训练批次正好包含来自每个类的 5000 张图像。

以下是数据集中的类，以及每个类的 10 张随机图像：

图 2 数据集

（三）使用的机器学习算法

1、随机森林 (Random Forest, RF):

依赖框架: scikit-learn

超参数调试:

n_estimators: 森林中树的数量。

max_depth: 树的最大深度。

2、残差网络（ResNet）:

依赖框架: PyTorch 。

超参数调试:

learning_rate: 优化算法的学习率。

batch_size: 每次迭代中用于计算损失和梯度的样本数量。

epochs: 迭代次数。

3、卷积神经网络 (Convolutional Neural Network, CNN):

依赖框架: PyTorch,

超参数调试:

learning_rate: 优化算法的学习率。

batch_size: 每次迭代中用于计算损失和梯度的样本数量。

optimizer: 优化算法，如 SGD, Adam 等。

二、数据库设计

（一）实体关系设计

用户（User）

用户 ID（User ID）：唯一标识每个用户。

注册日期（Registration Date）：用户注册账户的具体日期。

邮箱（Email）：用户的电子邮箱地址。

权限（Permission）：用户在系统中的权限级别，可能包括管理员、普通用户等。

用户名（Username）：用户在系统中的登录名。

性别（Gender）：用户的性别。

密码（Password）：用户账户的密码。

实验（Experiment）

实验 ID（Experiment ID）：唯一标识每个实验。

用户 ID（User ID）：与执行该实验的用户相关联。

预测结果（Prediction Result）：实验的预测输出。

实验时间（Experiment Time）：实验进行的具体时间。

模型（Model）

模型名（Model Name）：唯一标识每个模型。

训练（Training）：模型训练过程的相关信息。

运行（Running）：模型运行状态的相关信息。

评估结果（Evaluation Result）：模型性能的评估结果。

模型参数（Model Parameters）：模型训练时使用的参数。

模型保存路径（Model Save Path）：模型保存的位置。

关系：

用户与实验：一个用户可以进行多个实验，但每个实验是由单个用户执行的，因此它们之间是一对多的关系。

用户与模型：用户可以选择或创建多个模型，但每个模型可以由多个用户使用，因此它们之间是多对多的关系，可能需要一个关联表来处理这种关系。

实验与模型：每个实验使用一个模型，但一个模型可以用于多个实验，因此它们之间是多对一的关系。

绘制 E-R

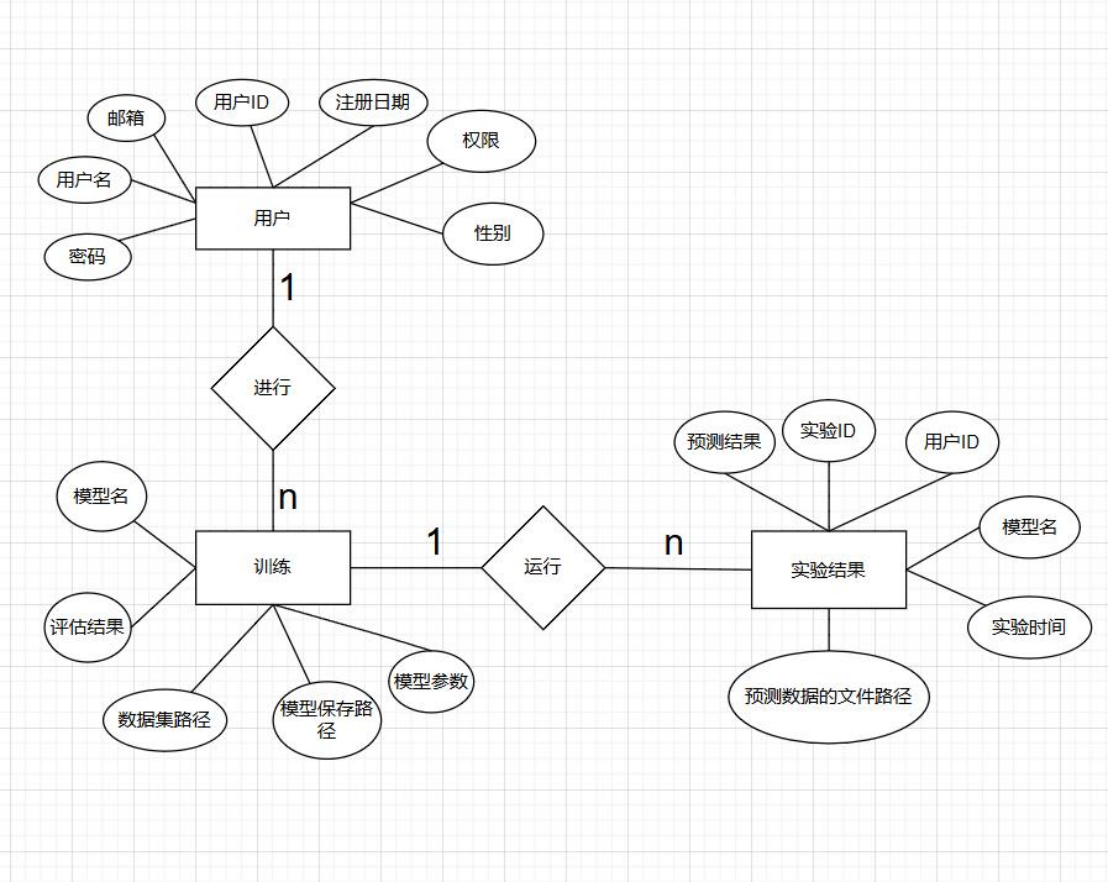

图 3 实体-关系（E-R）图

（二）数据库的实现

表 1 用户表（User）

字段名	类型	长度	备注	说明

字段名	类型	长度	备注	说明
UserId	int	-	Primary Key	用户 ID，主键，自动递增
Username	nvarchar	50	UNI	用户名，唯一约束
Email	nvarchar	100	UNI	邮箱，唯一约束
PasswordHash	nvarchar	255	-	加密后的密码
Gender	nvarchar	100	YES	性别，可以为空
IsAdmin	tinyint	1	0	是否是管理员，0 或 1
RegistrationDate	datetime	-	DEFAULT_GENERATED	注册日期，默认为当前时间

表 2 训练表（Train）

字段名	类型	长度	备注	说明
ModelName	nvarchar	255	Primary Key	模型名称，主键
ModelEval	nvarchar	255	YES	模型评估结果，可以为空
DataPath	nvarchar	255	YES	数据集路径，可以为空
ModelPath	nvarchar	255	YES	模型保存路径，可以为空
ModelParameters	json	-	YES	模型参数，以 JSON 格式存储，可以为空
UserId	int	-	MUL	用户 ID，外键关联用户表

表 3 实验结果表（ExperimentResults）

字段名	类型	长度	备注	说明
ResultId	int	-	Primary Key	实验结果 ID，主键，自动递增

字段名	类型	长度	备注	说明
UserId	int	-	MUL	用户 ID，外键关联用户表
ModelName	nvarchar	255	MUL	模型名称，可以为空
PredictionDataPath	nvarchar	255	YES	预测数据路径，可以为空
PredictionResult	text	-	YES	预测结果，可以为空
PredictionDate	datetime	-	DEFAULT_GENERATED	预测日期，默认为当前时间戳

三、功能模块实现

各功能模块的具体实现：

1、数据库模块

```
def connect_to_database(self):
    try:
        conn = mysql.connector.connect(
            host='127.0.0.1',
            database='machinelearn',
            user='root',
            password='123456'
        )
        return conn
    except mysql.connector.Error as err:
        print(f"数据库连接错误: {err}")
        return None

2 个用法
def close_database_connection(self, conn, cursor):
    if cursor:
        cursor.close()
    if conn and conn.is_connected():
        conn.close()
        print("数据库连接已关闭。")
```

2、登录模块

```
cursor = conn.cursor()
cursor.execute( operation: "SELECT UserID, PasswordHash, isadmin "
               "FROM Users "
               "WHERE UserID = %s",  params: (accountid,))
row = cursor.fetchone()

if row is not None:
    userID, password_hash, is_admin = row  # Unpack the values from the row
    if password == password_hash:
        if is_admin == 1:
            self.main_window = ManageWindow()
            self.main_window.show()
            self.close()
        else:
            print("登录成功")
            self.main_window = MainWindow(user_id=userID)  # Pass user_id to MainWindow
            self.main_window.show()
            self.close()  # Hide login window
    else:
        self.ui.stackedWidget.setCurrentIndex(2)  # Password incorrect
else:
    self.ui.stackedWidget.setCurrentIndex(1)  # User not found
```

如果是管理员就登录管理员界面否则登录用户界面

3、个人信息管理模块

```
cursor = conn.cursor()
cursor.execute(
     operation: "SELECT Email, Gender, PasswordHash, RegistrationDate, UserID, Username "
    "FROM Users "
    "WHERE UserID = %s",
     params: (self.user_id,))
row = cursor.fetchone()

if row:
    email, gender, password_hash, registration_date, user_id, username = row
    self.ui.lineEdit_Email.setText(email)
    self.ui.lineEdit_Gender.setText(gender)
    self.ui.lineEdit_PasswordHash.setText(password_hash)
    self.ui.lineEdit_RegistrationDate.setText(str(registration_date))
    self.ui.lineEdit_UserID.setText(str(user_id))
    self.ui.lineEdit_Username.setText(username)
else:
    print("用户未找到。")
```

读取文本框中个人信息并且更新到数据库中去，同样用用户 ID 作为限制

4、实验信息管理模块

```
    cursor = conn.cursor()
    cursor.execute(
        operation: "SELECT UserID,ResultID  ,ModelName, PredictionDate ,PredictionDataPath ,PredictionResult "
        "FROM Experimentresults "
        "WHERE UserID = %s",
        params: (self.user_id,))
    rows = cursor.fetchall()

    if rows:
        self.ui.tableWidget.setRowCount(len(rows))
        self.ui.tableWidget.setColumnCount(6)
        self.ui.tableWidget.setHorizontalHeaderLabels(['用户ID', '实验ID', '模型名', '实验时间', '预测数据路径', '实验结果'])

        for row_num, row_data in enumerate(rows):
            for col_num, data in enumerate(row_data):
                self.ui.tableWidget.setItem(row_num, col_num, QTableWidgetItem(str(data)))
    else:
        print("没有找到实验信息。")
```

从数据库中查询特定用户的实验结果，并将结果填充到表格控件中。首先，根据用户 ID 执行查询操作，然后获取所有匹配的记录。

```
selected_row = self.ui.tableWidget.currentRow()
if selected_row != -1:
    experiment_id = self.ui.tableWidget.item(selected_row, 1).text()  # 获取选中行的实验ID
    conn = self.connect_to_database()
    if conn:
        try:
            cursor = conn.cursor()
            cursor.execute( operation: "DELETE FROM Experimentresults WHERE ResultID = %s",
                            params: (experiment_id,))
            conn.commit()
            print(f"实验ID为 {experiment_id} 的实验信息删除成功。")
            self.load_experiments()  # 重新加载实验信息表格
        except mysql.connector.Error as err:
            print(f"数据库错误: {err}")
        finally:
            self.close_database_connection(conn, cursor)
else:
    QMessageBox.warning(self, "警告", "请先选择要删除的实验信息。", QMessageBox.Ok)
```

在选择要删除的实验信息后，系统会从数据库中执行删除操作，移除对应实验 ID 的实验结果记录，并随后重新加载实验信息表格以更新显示。

5、模型训练模块

```python
# Data preprocessing
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize( mean: (0.5, 0.5, 0.5), std: (0.5, 0.5, 0.5))
])

# Load CIFAR-10 dataset
trainset = CIFAR10(root=self.data_directory, train=True, download=True, transform=transform)
trainloader = DataLoader(trainset, batch_size=size_1, shuffle=True, num_workers=2)
testset = CIFAR10(root=self.data_directory, train=False, download=True, transform=transform)
testloader = DataLoader(testset, batch_size=size_1, shuffle=False, num_workers=2)

# Initialize model and optimizer
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = models.resnet18(pretrained=False, num_classes=10).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=IR_1, momentum=0.9)

# Training the model
num_epochs = epochs
for epoch in range(num_epochs):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data[0].to(device), data[1].to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

```python
    cur = conn.cursor()
    insert_stmt = (
        "INSERT INTO Train (ModelName, ModelEval, DataPath, ModelPath, ModelParameters, UserId) "
        "VALUES (%s, %s, %s, %s, %s, %s)"
    )

    cur.execute(insert_stmt, params: (
        'RESNET',  # 模型名称
        f"{accuracy :.2f}%",  # 模型评估结果
        self.data_directory,  # 数据集路径
        model_path,  # 模型保存路径
        json.dumps({'size_1': size_1, ' IR_1':  IR_1, 'epochs': epochs}),  # 模型参数信息转换为JSON格式
        self.user_id,  # 用户ID
    ))
    conn.commit()
    QMessageBox.information(self, "模型训练完成", "RESNET模型训练完成，信息已保存到数据库。")
except mysql.connector.Error as err:
    QMessageBox.critical(self, "数据库错误", f"发生错误：{err}")
```

加载 CIFAR-10 数据集，应用数据预处理（转换为张量并归一化）。初始化未预训练的 ResNet18 模型和优化器。在多个周期中训练模型，每批数据通过模型并计算损失，然后反向传播优化模型参数。训练期间，通过 UI 监控进度并允许用户取消训练过程。计算测试准确率，保存模型参数，并将结果存入数据库。其余同理

6、模型加载模块

```
if model_path.endswith('.pth'):
    if 'cnn' in model_path:
        self.model = CNN().to(device)
        self.model.load_state_dict(torch.load(model_path, map_location=device))
        self.model.eval()  # 设置为评估模式
        self.model_type = "cnn"
        print(f"Loaded CNN model from {model_path}")
    elif 'resnet' in model_path:
        self.model = models.resnet18(pretrained=False, num_classes=10).to(device)
        self.model.load_state_dict(torch.load(model_path, map_location=device))
        self.model.eval()  # 设置为评估模式
        self.model_type = "ResNet"
        print(f"Loaded ResNet model from {model_path}")
elif model_path.endswith('.pkl'):
    if 'decision_tree' in model_path:
        with open(model_path, 'rb') as f:
            self.model = pickle.load(f)
        self.model_type = "decision_tree"
        print(f"Loaded Decision Tree model from {model_path}")
```

通过文件对话框选择模型文件，根据文件后缀和文件名加载 CNN、ResNet 或随机森林模型。加载过程中，将模型状态字典或整个模型从文件中读取，并适配到当前设备（如 GPU 或 CPU）。加载完毕后，将模型设置为评估模式，

7、预测模块

```
if self.model_type in ["cnn", "ResNet"]:
    # 将模型移动到相同的设备上
    self.model.to(device)

    # 使用 CNN 或 ResNet 模型进行预测
    with torch.no_grad():  # 不需要计算梯度，节省内存和计算资源
        output = self.model(image)
        _, predicted = torch.max(output, 1)
    predicted_class = classes[predicted.item()]  # 假设classes是你的类标签列表
elif self.model_type == "random_forest":
    # 使用随机森林模型进行预测
    image_cpu = image.cpu()  # 将 CUDA 张量移动到 CPU 上
    image_numpy = image_cpu.view(-1).numpy()  # 将图像数据展平并转换为 NumPy 数组
    predicted_class = classes[self.model.predict([image_numpy])[0]]
else:
    print(f"Unsupported model type: {self.model_type}")
    return
```

```
# 查询 train 表中是否存在相应的模型名称记录
cur.execute( operation: "SELECT ModelName FROM Train WHERE ModelName = %s",  params: (model_name,))
result = cur.fetchone()
if result:
    model_name_found = result[0]
    print("Found model in train table:", model_name_found)
    # 这里可以根据需要继续处理查询结果

if result:
    # 构建插入语句
    insert_stmt = (
        "INSERT INTO experimentresults (modelname, predictiondatapath, predictionresult, userid) "
        "VALUES (%s, %s, %s, %s)"
    )

    cur.execute(insert_stmt,  params: (
        model_name,
        prediction_data_path,
        prediction_result,
        user_id,
    ))
    conn.commit()
    print("Prediction result saved to database.")
```

查图像文件是否存在，随后根据指定的模型类型（如 CNN、ResNet 等深度学习模型或随机森林等传统机器学习模型）对图像进行预处理（如调整大小、转换为张量并归一化）。处理后的图像被送入相应模型进行预测，最终输出预测的类别。并将实验数据（模型名、预测数据路径、预测结果）上传到数据库。

四、机器学习实验设计及结果

1、实验设计：

CNN（卷积神经网络）:

使用 PyTorch 框架。

可以采用 SGD 或 Adam 优化器。

超参数包括批量大小（size）、学习率（IR）。

ResNet（残差网络）:

特定为 ResNet18 模型，使用 PyTorch 框架。

使用 SGD 优化器，具有动量参数。

超参数包括批量大小（size_1）、初始学习率（IR_1）和训练周期数（epochs）。

随机森林 (Random Forest):

使用 scikit-learn 框架。

超参数包括估计器数量（n_estimators）、最大深度（max_depth）和随机状态（random_state）。

机器学习或深度学习框架：

PyTorch: 用于构建和训练 CNN 和 ResNet 模型。PyTorch 是一个开源的机器学习库，广泛用于计算机视觉和自然语言处理任务。

scikit-learn: 用于训练随机森林模型。scikit-learn 是一个简单有效的 Python 编程语言库，用于实现机器学习、数据挖掘和数据分析。

2、实验结果及分析：

在给定数据集的情况下，使用 0.03 的学习率、128 的批量大小和 10 个 epoch 的 ResNet 模型性能更好，其准确率达到了 71.62%，高于使用 0.01 的学习率、相同的批量大小和 5 个 epoch 的模型（准确率为 69.36%）。这表明在这种特定情况下，更高的学习率和更多的 epoch 训练次数可以提高模型的性能。

五、代码版本管理

表 4 Git 代码管理提交记录

序号	提交日期	提交说明	备注
1	6.19	新建仓库分支	
2	6.20	开发登录界面、个人信息管理界面、模型选择实验结果显示界面雏形	
3	6.21	登录界面连接数据库	
4	6.24	完善了用户信息的查询和修改和实验信息的查询	
5	6.25	实现了个人信息的管理和实验信息的管理	
7	6.26	实现了管理员模块	
8	6.27	实现模型训练和预测	
9	6.28	完善机器学习	
10	7.1	完善管理员	
11	7.3	完成全部	

图 4 Gitee 提交记录

六、自学知识

在不懈的实践与深入探索中，我全面掌握了多项编程技能与前沿技术，显著拓宽了我的技术视野与实战能力。首先，我精进了 PyQt5 这一强大的 Python GUI 框架，它不仅让我能够跨越平台界限，设计出功能丰富、界面友好的桌面应用程序，还深刻理解了事件驱动编程的精髓、信号与槽机制的精妙、以及窗口小部件的灵活应用与布局管理的艺术。这一过程不仅锤炼了我的编程技巧，也培养了我对用户体验设计的敏锐洞察。

此外，我积极拥抱了版本控制的现代实践，通过深入学习并应用 Gitee 平台，我熟练掌握了 Git，包括代码仓库的高效管理这技能不仅极大地提升了我个人编码工作的组织性与效率，也为我以后在团队协作中发挥打下了基础，促进了项目的顺利。

在 PyTorch 和 scikit-learn 的助力下，我深入学习了深度学习模型的构建与调优。从基础的卷积神经网络（CNN）到随机森林等算法，我不仅掌握了数据预处理、模型训练、性能评估与模型保存的完整流程。这一学习过程不

仅深化了我的编程功底，更激发了我对人工智能技术的浓厚兴趣与深刻理解。

七、总结与体会

本次课程设计任务不仅涵盖了深度学习模型的构建与评估，还包括了使用 PyQt5 设计图形用户界面（GUI），以增强用户交互性。任务的主要内容和目标包括： 深度学习模型开发：使用 PyTorch 框架，开发了包括 CNN 和随机森林在内的多种深度学习模型。 GUI 设计：利用 PyQt5 库设计了用户友好的图形界面，使得模型训练、评估和预测过程更加直观和易操作。 用户交互实现：通过 GUI，用户可以方便地输入参数、选择文件、启动训练、查看结果和调整模型设置。

最近的学习让我深刻体会到了数据库和 PyQt5 还有深度学习的实用性和魅力。通过学习数据库规范化，我认识到合理设计数据库结构对于数据的存储、查询和维护至关重要。而 PyQt5 则为我打开了图形用户界面设计的大门，让我能够创建出直观、易用的界面。

在学习的过程中，我感受到了知识的广度和深度，也遇到了不少挑战。但通过不断摸索和实践，我逐渐掌握了这些技能，并在实践中不断巩固和提升。这次学习让我深刻认识到，学习是一个不断积累和提升的过程。只有不断地学习、实践和创新，才能不断提高自己的能力和水平。我期待着将这些知识应用到未来的工作和项目中，为实现更好的成果贡献自己的力量。