1.17 工作报告

搭建深度学习模型预测 CO：

（1） 将提取出 ppg 数值特征输入网络，输入特征包含：

心率

心率变异率

交流分量与直流分量比值

二尖瓣波值与波谷距离与直流分量比值

ppg 波线下面积与矩形框住的 ppg 波面积比值

ppg 波形以二尖瓣波值为界前半部分面积与矩形框住 ppg 波形面积比值

ppg 波形以二尖瓣波值为界前半部分线下面积与后半部分线下面积比值

ppg 上升支线下面积与矩形面积比值

ppg 脉搏波动力学参数（一个周期内 ppg 波波峰与波谷垂直距离与对应这一段水平距离比值加上二尖瓣波值到周期中第二个波谷垂直距离以及相应这一段水平距离比值之和）

直流分量的自然对数值（对应高阶分量）

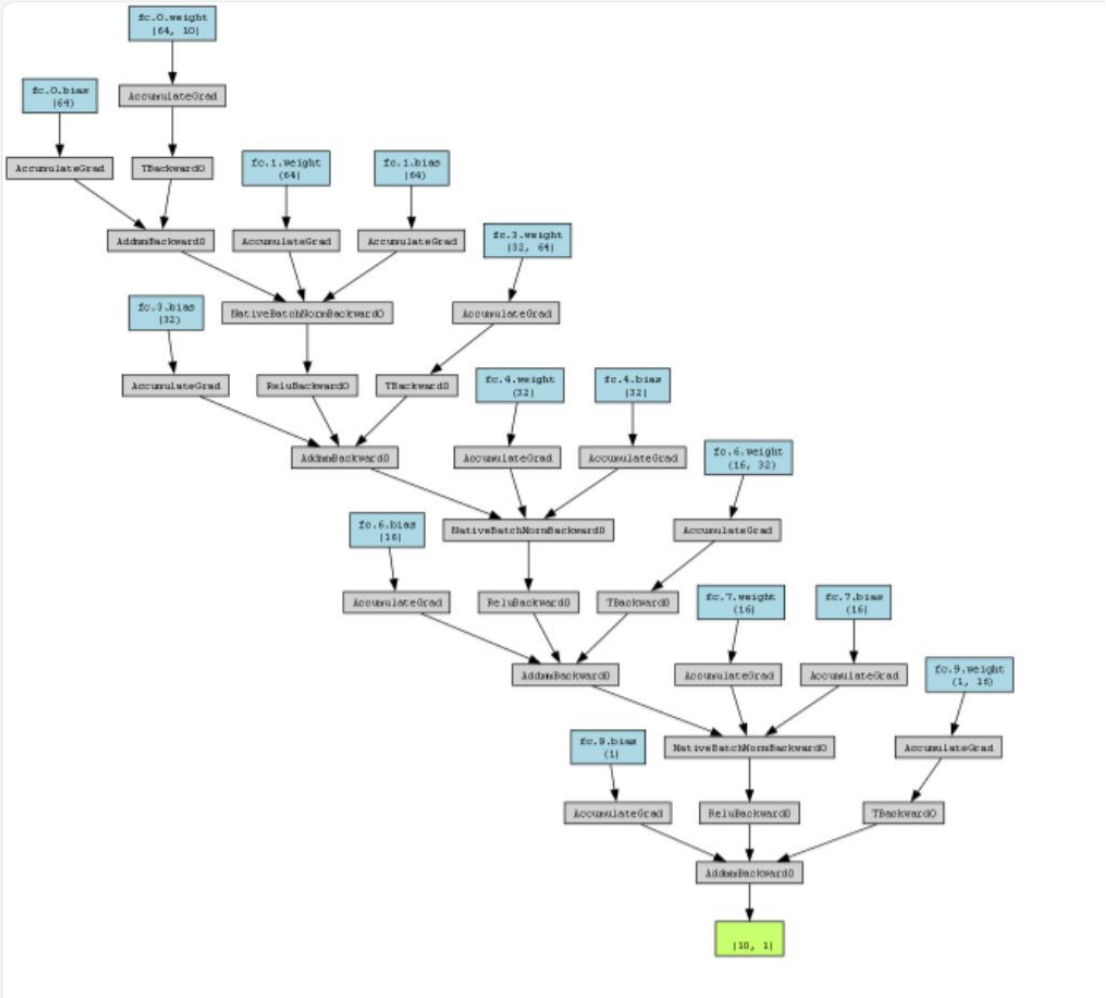
（2） 设置数据加载器 dataloader：遍历 data 文件夹中每一个 excel 文件，每一轮向训练中网络加载数据，同时将每一列数据归一化提高网络预测精度。

```
# 自定义数据集类
class CustomDataset(Dataset):
    def __init__(self, data_directory, columns_of_interest):
        self.data = []
        for data_file in os.listdir(data_directory):
            if data_file.endswith(".xlsx"):
                data_path = os.path.join(data_directory, data_file)
                df = pd.read_excel(data_path, usecols=columns_of_interest)
                df_normalized = normalize_dataframe(df)
                for _, row in df_normalized.iterrows():
                    numeric_data = row.drop('CO').astype(float)
                    co_label = row['CO'].astype(float)
                    self.data.append((numeric_data, co_label))

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        numeric_data, co_label = self.data[idx]
        numeric_data = torch.tensor(numeric_data.values, dtype=torch.float32)
        co_label = torch.tensor(co_label, dtype=torch.float32).unsqueeze(0)
        return numeric_data, co_label
```

（3） 设置 10 折交叉：首先将整个数据集均匀分割成 10 个子集，每个子集包含相同数量的数据点，然后进行 10 次训练和验证的迭代，在每次迭代中，从 10 个子集中挑选一个作为验证集，其余 9 个子集合并作为训练集。使用训练集来训练模型，并用独立的验证集来评估模型的性能，确保评估是在模型未曾见过的数据上进行。

```
dataset = CustomDataset(data_directory, columns_of_interest)
data_loader = DataLoader(dataset, batch_size=20, shuffle=True)
kf = KFold(n_splits=10)

# K折交叉
for fold, (train_idx, val_idx) in enumerate(kf.split(dataset)):
    print(f"Starting fold {fold + 1}")
    train_subset = torch.utils.data.Subset(dataset, train_idx)
    val_subset = torch.utils.data.Subset(dataset, val_idx)

    train_loader = DataLoader(train_subset, batch_size=20, shuffle=True)
    val_loader = DataLoader(val_subset, batch_size=20, shuffle=False)

    model = NumericNetwork(numeric_features=len(columns_of_interest) - 1)
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=0.0001)
```

（4） 网络结构体：



网络的具体结构为：

NumericNetwork ： 继承自 nn.Module PyTorch 架构中所有神经网络的基类

构造函数 __init__函数初始化网络结构：
numeric_features ： 是输入层的特征数量。

self.fc ： nn.Sequential 容器，定义了网络的层次结构。

nn.Linear(numeric_features, 64)：第一个线性层（全连接层），将输入的特征转换为64个特征。

nn.BatchNorm1d(64)：对64个特征进行批量归一化，加快训练速度，提高模型稳定性。

nn.ReLU()：激活函数，增加非线性，允许模型学习更复杂的关系。

nn.Linear(64, 32)：第二个线性层，进一步压缩特征到32个。

nn.BatchNorm1d(32)：对32个特征进行批量归一化。

nn.ReLU()：又一层激活函数。

nn.Linear(32, 16)：第三个线性层，将特征数量减少到16个。

nn.BatchNorm1d(16)：对16个特征进行批量归一化。

nn.ReLU()：一层激活函数。

nn.Linear(16, 1)：最后一个线性层，将特征数量减少到1个，用于输出。

self._initialize_weights()：一个自定义的方法，用于初始化网络权重，防止初始梯度设置不当造成梯度爆炸。

_initialize_weights ：这个方法通过遍历模型的所有模块（层）来初始化权重，也用于防止梯度爆炸。

forward 前向传播： 网络的前向传播函数。当网络输入数据时，数据会通过self.fc 定义的层次结构进行传播。并通过反向传播以均方损失函数MSE最小为目标更新梯度，最终，输出结果是一个单一的数值，由最后一个线性层产生。

```
class NumericNetwork(nn.Module):
    def __init__(self, numeric_features):
        super(NumericNetwork, self).__init__()
        self.fc = nn.Sequential(
            nn.Linear(numeric_features, 64),
            nn.BatchNorm1d(64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.BatchNorm1d(32),
            nn.ReLU(),
            nn.Linear(32, 16),
            nn.BatchNorm1d(16),
            nn.ReLU(),
            nn.Linear(16, 1)
        )
        self._initialize_weights()

    def forward(self, numeric_data):
        output = self.fc(numeric_data)
        return output

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Linear):
                nn.init.kaiming_normal_(m.weight, nonlinearity='relu')
                if m.bias is not None:
                    nn.init.constant_(m.bias, 0)
```

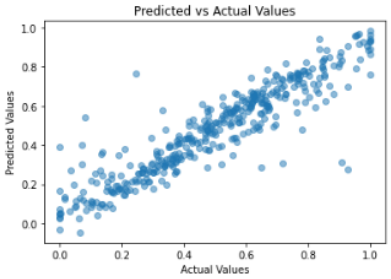
（5） 网络训练 每一轮训练 打印每一折数据集 MSE 均方损失函数 以及每一层更新的梯度检查是否出现梯度不稳定情况 同时打印每一轮输入网络的数据以及对应标签(CO 数值) 设置训练 5000 轮：

```
num_epochs = 5000
max_grad_norm = 0.01  # 设置梯度裁剪的最大范数 防止梯度爆炸
for epoch in range(num_epochs):
    model.train()
    epoch_loss = 0.0  #损失函数计数器
    for batch_idx, (numeric_data, co_label) in tqdm(enumerate(train_loader), total=len(train_loader)):

        optimizer.zero_grad()
        outputs = model(numeric_data)
        loss = criterion(outputs, co_label)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
        print(f"Fold {fold + 1}, Epoch {epoch + 1}, Batch {batch_idx + 1}, Loss: {loss.item()}")
        print("Input data:", numeric_data)
        print("Labels:", co_label)
    print(f"Fold {fold + 1}, Epoch {epoch + 1}, Average Epoch Loss: {epoch_loss / len(train_loader)}")


torch.save(model.state_dict(), f'model_weights_fold_{fold + 1}.pth')
```

（6） 训练完成后 保存模型 关闭梯度 将测试集数据输入网络进行预测 预测 R 方约 0.83

In [13]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score

#网络训练好之后 使用网络进行预测
#关闭梯度
#张量数据输入 Dataloader
all_data_loader = DataLoader(dataset, batch_size=20, shuffle=False)


model.eval()


all_predictions = []
all_actuals = []
with torch.no_grad():
    for numeric_data, co_label in all_data_loader:
        outputs = model(numeric_data)
        all_predictions.extend(outputs.view(-1).tolist())
        all_actuals.extend(co_label.view(-1).tolist())

#评估
r2 = r2_score(all_actuals, all_predictions)
print("R-squared:", r2)

#绘图
plt.scatter(all_actuals, all_predictions, alpha=0.5)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Predicted vs Actual Values")
plt.show()
```

```
R-squared: 0.8363430901486505
```



（7） 下一步工作计划：评估体内体外两个数据集预测 CO 的质量区别
搭建输入图像的网络遇到的问题：发现输出 R 方较低 且由于图像像素组成 数据量大 训练时间长 发现可能不适合用于预测 可能可以用于异常值标注